Version 30.0: Spring '14

# Database.com Apex Code Developer's Guide


Last updated: June 2, 2014

# Table of Contents

# GETTING STARTED

## Chapter 1

## Introduction

**In this chapter ...**



In this chapter, you'll learn about the Apex programming language, how it works, and when to use it.

# Introducing Apex

Salesforce.com 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 Database.com Automation and Service & Support, and build new custom applications based on particular business needs.

With the addition of Database.com to the Force.com platform, a multitenant cloud database service is provided to store data for custom mobile, social, and desktop applications. Database.com is the database for applications that are written in any language, and run on any platform or mobile device. Apex is an object-oriented programming language that enables you to add business logic and write triggers for your database on Database.com.

To learn more about Apex, see What is Apex?.

# What is Apex?

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on Database.com 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. Apex code can be initiated by Web service requests and from triggers on objects.

As a language, Apex is:

**Integrated**

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

- Data manipulation language (DML) calls, such as `INSERT`, `UPDATE`, and `DELETE`, that include built-in `DmlException` handling
- Inline Database.com Object Query Language (SOQL) and Database.com 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 Database.com. 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 Database.com, 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 Database.com.

**Multitenant aware**

Like the rest of Database.com, Apex runs in a multitenant environment. Consequently, the Apex runtime engine is designed to guard closely against runaway code, preventing them from monopolizing shared resources. Any code that violate these limits fail with easy-to-understand error messages.

**Automatically upgradeable**

Apex never needs to be rewritten when other parts of Database.com are upgraded. Because the compiled code is stored as metadata in the platform, it always gets automatically upgraded with the rest of the system.

**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. Database.com ensures that Apex code always work as expected by executing all unit tests stored in metadata 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.

# When Should I Use Apex?

Apex enables you to implement complex business processes and add custom functionality to your Database.com organization.

## Apex

Use Apex if you want to:

- Create Web 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 inserting a record, so that it occurs whenever the operation is executed.

## SOAP API

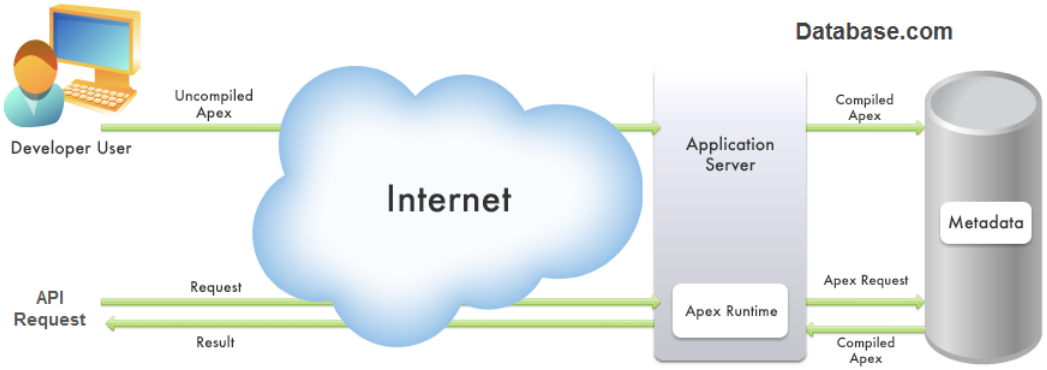
Use standard SOAP API calls if you want to add functionality to a composite application that processes only one type of record at a time and does not require any transactional control (such as setting a Savepoint or rolling back changes).

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

# How Does Apex Work?

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

**Figure 1: Apex is compiled, stored, and run entirely on Database.com.**




When a developer writes and saves Apex code to Database.com, the Database.com 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 Apex is executed, the Database.com application server retrieves the compiled instructions from the metadata and sends them through the runtime interpreter before returning the result.

## Developing Code in the Cloud

The Apex programming language is saved and runs in the cloud—the Database.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:

- 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 Database.com, 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 Database.com. 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 Spring '14 Release Notes to learn about new and changed Apex features in Spring '14.

### Past Releases

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

- 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

# Understanding Apex Core Concepts

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

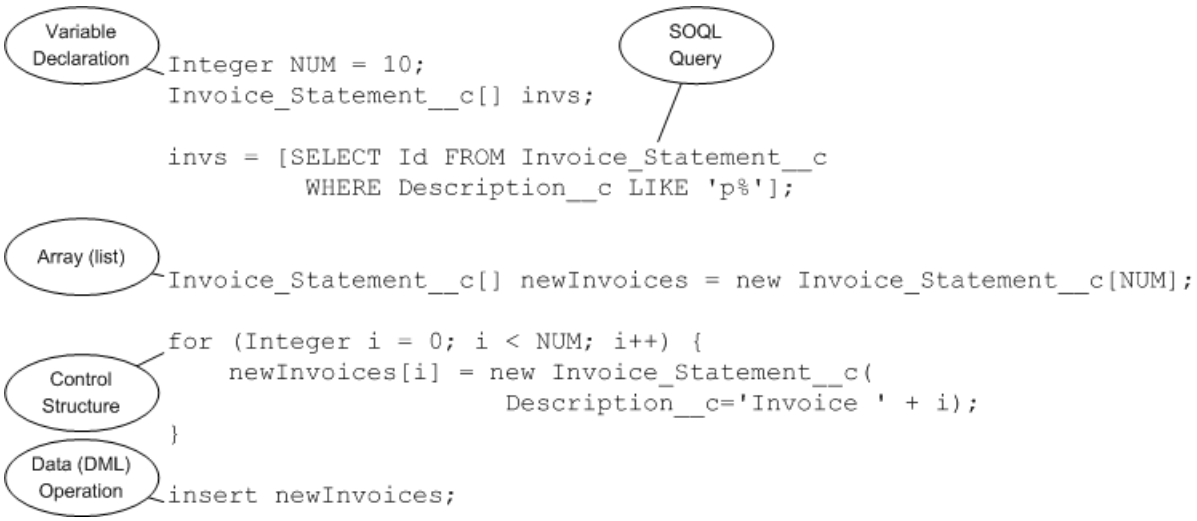



**Figure 2: Programming elements in Apex**

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

### Using Version Settings

In the Database.com user interface you can specify a version of the Salesforce.com 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.

### 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 Database.com, 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 invoice statement, which is also referred to as an sObject.
Invoice_Statement__c MyAcct = new Invoice_Statement__c(Description__c='Invoice 1');
```

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 25.

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 27.

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 27.

### 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 44.

### 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 44.

## Warehouse Objects for Code Samples

The code samples included in this guide are based on these custom objects:

- `Merchandise__c`
- `Invoice_Statement__c`
- `Line_Item__c`

A master-detail relationship relates `Invoice_Statement__c` with `Line_Item__c`. Similarly, `Merchandise__c` is related to `Line_Item__c` through another master-detail relationship.

You must create these objects in your development or test database organization before you can run the code samples. These objects are based on the Warehouse application in the *Force.com Workbook*. See the workbook for more information about how to create these objects and relationships.

# 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. Create a test database for your Database.com organization.
2. Write your Apex.
3. While writing Apex, you should also be writing tests.
4. Deploy your Apex to your Database.com production organization.

# Developing in a Test Database Organization

There are two types of organizations where you can run your Apex:

- A *production* organization: an organization that has live users accessing your data.
- A *test database* organization: an organization created on your production organization that is a copy of your production organization.

You can't develop Apex in your Database.com 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 a test database organization.

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

## Creating a Test Database Organization

To create or refresh a test database organization:

1. From Setup, click **Test Database** or **Data Management** > **Test Databases**.
2. Click **New Test Database**.
3. Enter a name and description for the test database. You can only change the name when you create or refresh a test database.

**Tip:** We recommend that you choose a name that:

- Reflects the purpose of this test database, such as "QA."
- Has few characters because Database.com automatically appends the test database name to usernames on user records in the test database environment. Names with fewer characters make test database logins easier to type.

4. Select the type of test database you want.

**QA Database**

QA databases are intended for coding and testing by a single developer. Multiple users can log into a single QA database, but their primary purpose is to provide an environment in which changes under active development can be isolated until they're ready to be shared. QA databases copy all application and configuration information to the test database. QA databases are limited to 200 MB of test or sample data, which is enough for many development and testing tasks. You can refresh a QA database once per day.

**Staging Database**

Staging databases copy your entire production organization and all its data, including custom object records. You can refresh a staging database every 29 days.

**Note:** If you don't see a test database option or need licenses for more test databases, contact salesforce.com to order test databases for your organization.

If you have reduced the number of test databases you purchased, but you still have more test databases of a specific type than allowed, you will be required to match your test databases to the number of test databases that you purchased. For example, if you have two Staging test databases but purchased only one, you cannot refresh your Staging test database as a Staging test database. Instead, you must choose one Staging test database to convert to a smaller test database, such as a QA test database.

5. Select the data you want to include in your test database (you have this option for a Staging test database).

   For a Staging test database, choose how much object history to copy. Object history is the field history tracking of custom objects. You can copy from 0 to 180 days of object history, in 30–day increments. The default value is 0 days. Decreasing the amount of data you copy can significantly speed up test database copy time.

   You can choose to include **Template-based** data for a Staging test database. For this option, you need to have already created a test database template. Then you can pick the template from a list of templates you've created. For more information, see "Creating Test Database Templates" in the Salesforce Help.

6. Click **Create**.

   The process may take several minutes, hours, or even days, depending on the size of your organization.

**Tip:** Try to limit changes in your production organization while the test database copy proceeds.

# 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 Database.com organization.

To open the Developer Console in Database.com 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.com to support its users and partners but isn't considered part of our services for purposes of the salesforce.com 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 316 and SOAP API and SOAP Headers for Apex on page 1073.

### Code Editor in the Database.com User Interface

The Database.com 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 Database.com 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 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 Database.com user interface in a Database.com 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, 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 to a production organization, every unit test in your organization namespace is executed.
  - 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 295.

## Deploying Apex to a Database.com 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 Database.com production organization.

To deploy Apex from a local project in the Force.com IDE to a Database.com organization, use the Force.com Component Deployment Wizard. For more information about the Force.com IDE, see `http://wiki.developerforce.com/index.php/Force.com_IDE`.

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

For more information and for additional deployment options, see Deploying Apex on page 316.

# Chapter 3

# Apex Quick Start

Once you have a test database 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.

## 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.



### Creating a Custom Object

Prerequisites:

A Database.com account in a test database Database.com organization.

For more information about creating a test database organization, see “Test Database Overview” in the Database.com online help.

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

1. Log into your test database 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 **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 Database.com account in a test database Database.com 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 48.

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 on page 55.

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.

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. 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 33.

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 Database.com account in a test database Database.com 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 platform databaseDatabase.com. 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 Database.com account in a test database Database.com 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, 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 to a production organization, every unit test in your organization namespace is executed.
  - 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 73 and Understanding 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 Database.com account in a test database Database.com organization.
- The HelloWorldTestClass Apex test class.
- A deployment connection between the test database and production organizations that allows inbound change sets to be received by the production organization. See "Change Sets Overview" in the Database.com 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.

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.

# WRITING APEX

# 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 22)
- An sObject, either as a generic sObject or as a specific sObject, such as Invoice_Statement__c (see sObject Types on page 89 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 25)
  - A set of primitives (see Sets on page 27)
  - A map from a primitive to a primitive, sObject, or collection (see Maps on page 27)
- A typed list of values, also known as an *enum* (see Enums on page 29)
- Objects created from user-defined Apex classes (see Classes, Objects, and Interfaces on page 48)
- 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 268.

# 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 datatype 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:<br>`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.<br>You cannot manipulate a Date value, such as add days, merely by adding a number to a Date variable. You must use the Date methods instead. |

| 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.<br><br>You cannot manipulate a Datetime value, such as add minutes, merely by adding a number to a Datetime variable. You must use the Datetime methods instead. |
| Decimal | A number that includes a decimal point. Decimal is an arbitrary precision number. Currency fields are automatically assigned the type Decimal.<br><br>If you do not explicitly set the *scale*, that is, the number of decimal places, for a Decimal using the `setScale` method, the scale is determined by the item from which the Decimal is created.<br><br>• If the Decimal is created as part of a query, the scale is based on the scale of the field returned from the query.<br>• If the Decimal is created from a String, the scale is the number of characters after the decimal point of the String.<br>• If the Decimal is created from a non-decimal number, the scale is determined by converting the number to a String and then 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:<br><br>`Double d=3.14159;`<br><br>Note that scientific notation (e) for Doubles is not supported. |
| ID | Any valid 18-character Force.com record identifier. For example:<br><br>`ID id='00300000003T2PGAA0';`<br><br>Note that if you set `ID` to a 15-character value, Apex automatically 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:<br><br>`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 datatype when you need a range of values wider than those provided by Integer. For example:<br><br>`Long l = 2147483648L;` |
| Object | Any data type that is supported in Apex—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:<br>`Object obj = 10;`<br>`// Cast the object to an integer.`<br>`Integer i = (Integer)obj;`<br>`System.assertEquals(10, i);`<br>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.<br>`Object obj = new MyApexClass();`<br>`// Cast the object to the MyApexClass custom type.`<br>`MyApexClass mc = (MyApexClass)obj;`<br>`// Access a method on the user-defined class.`<br>`mc.someClassMethod();` |
| String | Any set of characters surrounded by single quotes. For example,<br>`String s = 'The quick brown fox jumped over the lazy dog.';`<br>**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.<br>**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.<br>Conversely, Strings in Apex can be `null` or empty, and can include leading and trailing whitespace (such as might be used to construct a message).<br>**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).<br>**Comparison Operators**: Unlike Java, Apex Strings support use of the comparison operators ==, !=, <, <=, >, and >=. Since 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 33.<br>**String Methods**: As in Java, Strings can be manipulated with a number of standard methods. See String Class for information.<br>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 Database.com 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 Database.com*.

# 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 847.

### 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));
```

## 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 928.

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. Do not rely on the order in which set results are returned. The order of objects returned by sets may change without warning.

## 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 861.

### 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.
- Do not rely on the order in which map results are returned. The order of objects returned by maps may change without warning. 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 Database.com WSDLs and Client Authentication Certificates" in the Database.com 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 Database.com 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.

- `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:

  ```
  Merchandise__c m1;
  MERCHANDISE__C m2;
  ```

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

```
Merchandise__c[] merchItems = [sELect ID From MErchanDIse__c where nAme = 'Pencils'];
```

**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 Database.com 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 68.

## 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
- Understanding Expression Operators
- Understanding Operator Precedence
- Expanding sObject and List Expressions

- Using Comments

## 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 Invoice_Statement__c(<field_initializers>)
new Integer[<n>]
new Invoice_Statement__c[]{<elements>}
new List<Invoice_Statement__c>()
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]
myInvoice.Description__c
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 line items associated with a particular invoice statement). 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:

```
Invoice_Statement__c[] aa = [SELECT Id, Description__c FROM Invoice_Statement__c
                             WHERE Description__c ='some text'];
Integer i = [SELECT COUNT() FROM Merchandise__c WHERE Description__c ='Pencils'];
List<List<SObject>> searchList = [FIND 'map*' IN ALL FIELDS
                                  RETURNING Merchandise__c (Id, Description__c),
                                  Invoice_Statement__c, Line_Item__c];
```

  For information, see SOQL and SOSL Queries on page 111.
- 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.<br>Note:<br>• This operator exhibits "short-circuiting" behavior, which means y is evaluated only if x is false.<br>• 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.<br>Note:<br>• This operator exhibits "short-circuiting" behavior, which means y is evaluated only if x is true.<br>• 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. |
| >>>= | 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. |

| Operator | Syntax | Description |
|---|---|---|
| `? :` | `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.<br>Note:<br>• `&&` has precedence over `\|\|`<br>• This operator exhibits "short-circuiting" behavior, which means `y` is evaluated only if `x` is true.<br>• `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.<br>Note:<br>• `&&` has precedence over `\|\|`<br>• This operator exhibits "short-circuiting" behavior, which means `y` is evaluated only if `x` is false.<br>• `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.<br>Note:<br>• Unlike Java, == in Apex compares object value equality, not reference equality, except for user-defined types. Consequently:<br>◊ String comparison using == is case insensitive<br>◊ ID comparison using == is case sensitive, and does not distinguish between 15-character and 18-character formats<br>◊ 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.<br>• 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.<br>• For records, every field must have the same value for == to evaluate to true.<br>• `x` or `y` can be the literal `null`.<br>• The comparison of any two values can never result in `null`.<br>• 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 |

| Operator | Syntax | Description |
|---|---|---|
| | | 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.<br>Note:<br>• 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`.<br>• If `x` or `y` equal `null` and are Integers, Doubles, Dates, or Datetimes, the expression is false.<br>• A non-`null` String or ID value is always greater than a `null` value.<br>• If `x` and `y` are IDs, they must reference the same type of object. Otherwise, a runtime error results.<br>• If `x` or `y` is an ID and the other value is a String, the String value is validated and treated as an ID.<br>• `x` and `y` cannot be Booleans.<br>• The comparison of two strings is performed according to the locale of the context user. |
| > | `x > y` | **Greater than operator**. If `x` is greater than `y`, the expression evaluates to true. Otherwise, the expression evaluates to false.<br>Note:<br>• The comparison of any two values can never result in `null`.<br>• If `x` or `y` equal `null` and are Integers, Doubles, Dates, or Datetimes, the expression is false.<br>• A non-`null` String or ID value is always greater than a `null` value.<br>• If `x` and `y` are IDs, they must reference the same type of object. Otherwise, a runtime error results.<br>• If `x` or `y` is an ID and the other value is a String, the String value is validated and treated as an ID.<br>• `x` and `y` cannot be Booleans.<br>• The comparison of two strings is performed according to the locale of the context user. |

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

| Operator | Syntax | Description |
|---|---|---|
| | | • 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:<br>• 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.<br>• If `x` is a Date and `y` is an Integer, returns a new Date that is incremented by the specified number of days.<br>• 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.<br>• 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:<br>• If `x` and `y` are Integers or Doubles, subtracts the value of `x` from the value of `y`. If a Double is used, the result is a Double.<br>• If `x` is a Date and `y` is an Integer, returns a new Date that is decremented by the specified number of days.<br>• 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. |
| `++` | `x++`<br>`++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--`<br>`--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. |

| Operator | Syntax | Description |
|---|---|---|
| `&` | `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. |
| `<<` | `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 |
| 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;
Invoice_Statement__c a = new Invoice_Statement__c();
Invoice_Statement__c[] invs = [SELECT Id FROM Invoice_Statement__c];
```

- A de-referenced list element. For example:

```
ints[0] = 1;
Invoice_Statement__c[0].Description__c = 'description';
```

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

```
Invoice_Statement__c a = new Invoice_Statement__c();

// IDs cannot be set manually
// a.Id = 'a00900000013R8Q';  This code is invalid!

// 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!

// Create a merchandise item to use for the line item
Merchandise__c m = new Merchandise__c(
   Name='Pencils',
   Description__c='Durable pencils',
   Price__c=1.25,
   Total_Inventory__c=100);
insert m;
```

```
// Since the invoice a has been inserted, it is now possible to
// create a new line item that is related to it
Line_Item__c li = new Line_Item__c(
                                  Name='Two pencils',
                                  Units_Sold__c=2,
                                  Unit_Price__c=5,
                                  Merchandise__c = m.id,
                                  Invoice_Statement__c=a.Id);

insert li;
Line_Item__c li2 = [SELECT Id,Invoice_Statement__r.Description__c
                    FROM Line_Item__c WHERE Id=:li.Id];
// Notice that you can write to an invoice statement field directly
// through the relationship field on the line item
li2.Invoice_Statement__r.Description__c = 'new description';
```

Assignment is always done by reference. For example:

```
Invoice_Statement__c a = new Invoice_Statement__c();
Invoice_Statement__c b;
Invoice_Statement__c[] c = new Invoice_Statement__c[]{};
a.Description__c = 'Invoice 1';
b = a;
c.add(a);

// These asserts should now be true. You can reference the data
// originally allocated to invoice a through invoice b and invoice list c.
System.assertEquals(b.Description__c, 'Invoice 1');
System.assertEquals(c[0].Description__c, 'Invoice 1');
```

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

```
Invoice_Statement__c[] a = new Invoice_Statement__c[]{new Invoice_Statement__c()};
Invoice_Statement__c[] b = a;
a[0].Description__c = 'Invoice 1';
System.assert(b[0].Description__c == 'Invoice 1');
```

In addition to =, other valid assignment operators include +=, *=, /=, |=, &=, ++, and --. See Understanding Expression Operators on page 33.

## 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

**In this chapter ...**



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.

# 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

## 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 Database.com.

Use the following syntax for defining classes:

```
private | public | global
[virtual | abstract | with sharing | without sharing | (none)]
class ClassName [implements InterfaceNameList | (none)] [extends ClassName | (none)]
{
```

```
// 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 71.
- 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.

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 65.

For more information about method and variable access modifiers, see Access Modifiers on page 54.

## 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 DML Statements on page 324.
- 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 41.

  **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 or Boolean. 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 49.

**Note:** Interface methods have no access modifiers. They are always global. For more information, see Understanding Interfaces on page 65.

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

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

- Static methods, variables, or initialization code are associated with a class, and are only allowed in outer classes. When you declare a method or variable as `static`, it's initialized only once when a class is loaded.
- Instance methods, member variables, and initialization code are associated with a particular object and have no definition modifier. When you declare instance methods, member variables, or initialization code, an instance of that item is created with every object instantiated from the class.
- Local variables are associated with the block of code in which they are declared. All local variables should be initialized before they are used.

The following is an example of 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 only use static methods and variables with outer classes. Inner classes have no static methods or variables. A static method or variable does not require an instance of the class in order to run.

All static member variables in a class are initialized before any object of the class is created. This includes any static initialization code blocks. All of these are run in the order in which they appear in the class.

Static methods are generally used as utility methods and never depend on a particular instance member variable value. Because a static method is only associated with a class, it cannot access any instance member variable values of its class.

Static variables are only static within the scope of the request. They're not static across the server, or across the entire organization.

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

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

Static variables defined in a trigger don't retain their values between different trigger contexts within the same transaction, for example, between before insert and after insert invocations. Define the static variables in a class instead so that the trigger can access these class member variables and check their static values.

Class static variables cannot be accessed through an instance of that class. So if class `C` has a static variable `S`, and `x` is an instance of `C`, then `x.S` is not a legal expression.

The same is true for instance methods: if `M()` is a static method then `x.M()` is not legal. Instead, your code should refer to those static identifiers using the class: `C.S` and `C.M()`.

If a local variable is named the same as the class name, these static methods and variables are hidden.

Inner classes behave like static Java inner classes, but do not require the `static` keyword. Inner classes can have instance member variables like outer classes, but there is no implicit pointer to an instance of the outer class (using the `this` keyword).

**Note:** For Apex saved using Salesforce.com API version 20.0 or earlier, if an API call causes a trigger to fire, the chunk of 200 records to process is further split into chunks of 100 records. For Apex saved using Salesforce.com API version 21.0 and later, no further splits of API chunks occur. Note that static variable values are reset between API batches, but governor limits are not. Do not use static variables to track state information between API batches.

## Using Instance Methods and Variables

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

Suppose you wanted to have a class that collects two dimensional points and plot them on a graph. The following skeleton class illustrates this, making use of 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 every time an object is instantiated from that class. These code blocks run before the constructor.

If you do not want to write your own constructor for a class, you can use an instance initialization code block to initialize instance variables. However, most of the time you should either give the variable a default value or use the body of a constructor to do initialization and not use instance initialization code.

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, the same as in Java.

You can use static initialization code to initialize static final variables and to declare any 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 54.
- *return_type* is the type of the property, such as Integer, Double, sObject, and so on. For more information, see Data Types on page 22.
- *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 59.

### 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 77.

## 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 Database.com 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<Invoice_Statement__c>{

   List<Invoice_Statement__c>
      invoices {get; set;}
   Integer i {get; set;}

 public CustomIterable(){
  invoices =
  [SELECT Id, Description__c
  FROM Invoice_Statement__c
  WHERE Description__c = 'false'];

       i = 0;
   }

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

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

The following calls the above code:

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

The following is a batch job that uses an iterator:

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

```
        for(Invoice_Statement__c a : scope){
            a.Description__c = 'New description';
            invsToUpdate.add(a);
        }
        update invsToUpdate;
    }
    global void finish(Database.batchableContext info){
    }
}
```

# Keywords

Apex has the following keywords available:

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

## 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 runtime 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.
}
```

## 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. For example:

```
Transient Integer currentTotal;
```

You can also use the `transient` keyword in Apex classes that are serializable, namely 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.

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:

- 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.

**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 156.

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:

```
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 if the class is called by a 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 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:

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

## `Future` Annotation

Use the `future` annotation to identify methods that are executed asynchronously. When you specify `future`, the method executes when Database.com 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.
- 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.

## `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 a utility method for test data creation:

```
@isTest
public class TestUtil {

   public static void createTestData() {
      // Create some test invoices
   }

}
```

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.com 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.com 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.com 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 301.

#### 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 merchandise item.
    // It also creates and accesses a new test merchandise item.
    static testmethod void myTestMethod1() {
        // Query an existing merchandise item in the organization.
        Merchandise__c m = [SELECT Id, Price__c, Total_Inventory__c, Description__c
                            FROM Merchandise__c WHERE Name='Pencils' LIMIT 1];
        System.assert(m != null);

        // Create a test merchandise item based on the queried merchandise item.
        Merchandise__c testMerchandise = m.clone();
        testMerchandise.Name = 'Test Pencil';
        insert testMerchandise;

        // Query the test merchandise that was inserted.
        Merchandise__c testMerchandise2 = [SELECT Id, Price__c, Total_Inventory__c
                            FROM Merchandise__c WHERE Name='Test Pencil' LIMIT 1];
        System.assert(testMerchandise2 != 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 invoice that is created here.
        Invoice_Statement__c inv = new Invoice_Statement__c(
                                   Description__c='Invoice 1');
        insert inv;
        // Access the invoice that was just created.
        Invoice_Statement__c insertedInv = [SELECT Id,Description__C
                                FROM Invoice_Statement__c
                                WHERE Description__c='Invoice 1'];
        System.assert(insertedInv != null);
    }
}
```

## `ReadOnly` Annotation

The `@ReadOnly` annotation allows you to perform unrestricted queries against the 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`, and calls to methods annotated with `@future`.

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.

## `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`

### `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);
   }
}
```

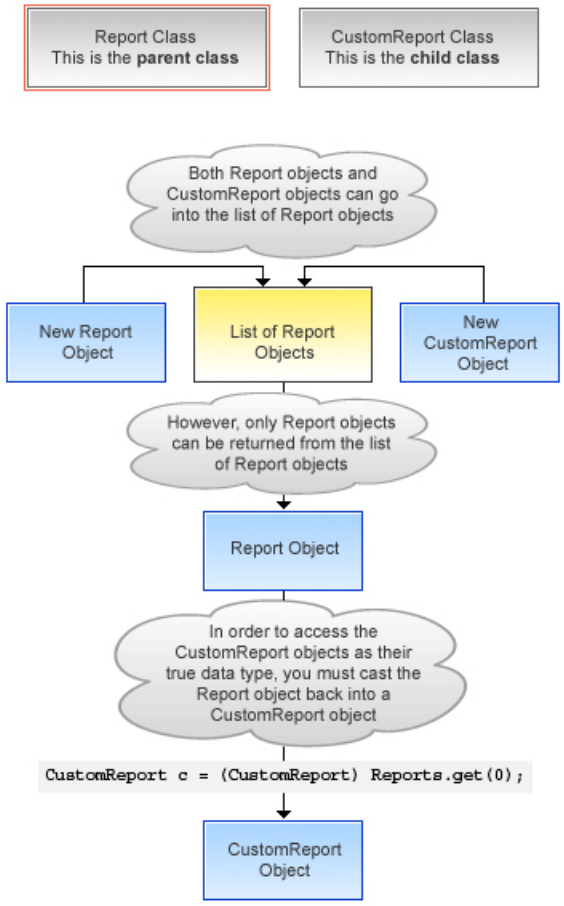



**Figure 3: Casting Example**

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 68.

## 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 only for the value of a map, not for the key. Likewise, you cannot 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.

## 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.
- Inner classes behave like static Java inner classes, but do not require the `static` keyword. Inner classes can have instance member variables like outer classes, 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 Database.com:

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. Use the default values for all versions. This associates the class with the most recent version of Apex and the API. 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 228.

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. 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 Apex, 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.

## 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 Database.com application supports the use of *namespace prefixes*.

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.

## 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'));
```

## 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 { Invoice_Statement__c a; }
}
```

Then the following expressions are all legal:

```
c.c1.c2.a.name
c.c1.c2.a.owner.lastName.toLowerCase()
```

## 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.com API version.

Typically, you reference the latest Salesforce.com API version. If you save an Apex class or trigger without specifying the Salesforce.com API version, the class or trigger is associated with the latest installed version by default.

## Setting the Database.com API Version for Classes and Triggers

To set the Salesforce.com 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.com 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.com API version setting, the fields in the objects are controlled by the version settings of C2.

# 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 Database.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 Salesforce Object Query Language (SOQL), 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 Database.com. 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:

```
Invoice_Statement__c co = new Invoice_Statement__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 Invoice_Statement__c();
```

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 an invoice statement
Invoice_Statement__c a = (Invoice_Statement__c)s;
// The following generates a runtime error
Merchandise__c c = (Merchandise__c)s;
```

Because sObjects work like objects, you can also have the following:

```
Object obj = s;
// and
a = (Invoice_Statement__c)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:

```
Invoice_Statement__c a = new Invoice_Statement__c();
```

Developers can also specify initial field values with comma-separated `name = value` pairs when instantiating a new sObject. For example:

```
Invoice_Statement__c a = new Invoice_Statement__c(
                  Description__c = 'New invoice');
```

For information on accessing existing sObjects from Database.com, see "SOQL and SOSL Queries" in the *Database.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. Database.com 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 25.

## 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 Database.com online help.

## Accessing sObject Fields

As in Java, sObject fields can be accessed or changed with simple dot notation. For example:

```
Invoice_Statement__c a = new Invoice_Statement__c();
a.Description__c = 'Invoice 1';    // Access the description field and assign it a value
```

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 Invoice_Statement__c, you can retrieve only the `Id` field using dot notation. You can set the `Id` field for Apex code saved using Salesforce.com 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.

```
Invoice_Statement__c a = new Invoice_Statement__c(
           Description__c = 'Invoice 1');
insert a;
sObject s = [SELECT Id, Description__c
             FROM Invoice_Statement__c
             WHERE Description__c = 'Invoice 1'
             LIMIT 1];
// This is allowed
ID id = s.Id;
// The following line results in an error when you try to save
String x = s.Description__c;
// This line results in an error when you try to save using API version 26.0 or earlier
s.Id = [SELECT Id
        FROM Invoice_Statement__c
        WHERE Description__c = 'Invoice 1'
        LIMIT 1].Id;
```

If you want to perform operations on an sObject, it is recommended that you first convert it into a specific object. For example:

```
Invoice_Statement__c a = new Invoice_Statement__c(
                Description__c = 'Invoice 1');
insert a;
sObject s = [SELECT Id, Description__c FROM Invoice_Statement__c
             WHERE Description__c = 'Invoice 1' LIMIT 1];
ID id = s.ID;
Invoice_Statement__c myInvoice = (Invoice_Statement__c)s;
myInvoice.Description__c = 'Updated description';
update myInvoice;
```

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 into a `Merchandise__c` or `Invoice_Statement__c` object, you can modify its fields accordingly:

```
static testmethod void testFields() {

    List<Merchandise__c> merchandise;
    List<Invoice_Statement__c> invoices;

    List<List<sObject>> results = [FIND 'pencil'
                    IN ALL FIELDS
                    RETURNING Merchandise__c(Id, Description__c, Price__c),
                    Invoice_Statement__c(Id, Description__c, Status__c)];
    sObject[] records = ((List<sObject>)results[0]);
    system.debug('Records returned: ' + records.size());
```

```
    if (!records.isEmpty()) {
        for (Integer i = 0; i < records.size(); i++) {
          sObject record = records[i];
          if (record.getSObjectType() == Merchandise__c.sObjectType) {
            merchandise.add((Merchandise__c) record);
          } else if (record.getSObjectType() == Invoice_Statement__c.sObjectType){
            invoices.add((Invoice_Statement__c) 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 Database.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 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 invoice statement sObject and assign it to a variable.

```
Invoice_Statement__c s = new Invoice_Statement__c(Description__c='Invoice Example');
```

In the previous example, the invoice statement referenced by the variable `s` exists in memory with the `Description__c` field. However, it is not persisted yet to the Database.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 invoice using the `insert` statement.

```
Invoice_Statement__c s = new Invoice_Statement__c(Description__c='Invoice Example');
insert s;
```

Also, you can use DML to modify records that have already been inserted. Among the operations you can perform are record updates, deletions, and restoring records from the Recycle Bin. 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 field on the sObject representation of this record in memory, and then persisting this change to the database.

```
// Query existing account.
Invoice_Statement__c inv =
                [SELECT Description__c
                FROM Invoice_Statement__c
                WHERE Description__c='Invoice Example'
                LIMIT 1];

// Write the old values the debug log before updating them.
System.debug(Invoice description before update: ' + inv.Description__c); // Description is
 Invoice Example

// Modify the description field on the sObject.
inv.Description__c = 'New description';

// Persist the change.
update inv;

// Get a new copy of the account from the database with the two fields.
Invoice_Statement__c inv2 =
                [SELECT Description__c
                FROM Invoice_Statement__c
                WHERE Description__c='New description'
                LIMIT 1];

// Verify that updated field value was persisted.
System.assertEquals('New description', inv2.Description__c);
```

# 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<Invoice_Statement__c> invList = new List<Invoice_Statement__c>();
invList.add(new Invoice_Statement__c(Description__c='Acme1'));
invList.add(new Invoice_Statement__c(Description__c='Acme2'));

// DML statement
insert invList;
```

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<Invoice_Statement__c> invList = new List<Invoice_Statement__c>();
invList.add(new Invoice_Statement__c(Description__c='Acme1'));
invList.add(new Invoice_Statement__c(Description__c='Acme2'));

// DML statement
Database.SaveResult[] sr = Database.insert(invList, 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 record with 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('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.

## 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 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 an exception that can't be caught for exceeding the DML statement limit of 150.

```
for(Line_Item__c badLi : liList) {
    if (badLi.Units_Sold__c > 10) {
        badLi.Description__c = 'New description';
    }
    // Not a good practice since governor limits might be hit.
    update badLi;
}
```

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 line items. This counts as one DML statement, which is far below the limit of 150.

```
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);
    }
}

// One DML call for the entire list of line items
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 merchandise items and update 50 merchandise items 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. The only exception is when a DML operation is called in a class defined with the `with sharing` keywords, the current user's sharing rules are taken into account.

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 merchandise records and update an existing merchandise record. First, it creates three Merchandise__c sObjects and adds them to a list. It then performs a bulk insertion by inserting the list of merchandise records using one `insert` statement. Next, it queries the second merchandise record, updates the price, and calls the `update` statement to persist the change in the database.

```
List<Merchandise__c> merList = new List<Merchandise__c>();
for(Integer i=0;i<3;i++) {
    Merchandise__c mer = new Merchandise__c(
        Name='Pen' + i,
        Description__c='Black pens ' + i,
        Price__c=1.25,
        Total_Inventory__c=100);
```

```
    merList.add(mer);
}

Merchandise__c merToUpdate;
try {
    insert merList;

    // Update merchandise Pen2.
    merToUpdate =
        [SELECT Price__c FROM Merchandise__c WHERE Name='Pen2' LIMIT 1];
    // Update the price.
    merToUpdate.Price__c = 2;
    // Make the update call.
    update merToUpdate;
} catch(DmlException e) {
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}

// Verify that the billing city was updated to New York.
Merchandise__c afterUpdate =
    [SELECT Price__c FROM Merchandise__c WHERE Id=:merToUpdate.Id];
System.assertEquals(2, afterUpdate.Price__c);
```

### 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 line item, you can specify the line item's related merchandise record by setting the value of the `Merchandise__c` field.

This example shows how to add a line item to an invoice statement and a merchandise record (the related records) by setting the `Merchandise__c` and the `Invoice_Statement__c` fields on the line item. Line_Item__c is linked to Merchandise__c and Invoice_Statement__c through two separate master-detail relationships.

```
try {
    // Create prerequisite records.
    Merchandise__c m = new Merchandise__c(
        Name='Pen',
        Description__c='Black pens',
        Price__c=1.25,
        Total_Inventory__c=100);
    insert m;

    Invoice_Statement__c inv = new Invoice_Statement__c(
        Description__c='Invoice 1');
    insert inv;

    // Create the line item and relate it to the merchandise and invoice statement.
    Line_Item__c li = new Line_Item__c(
        Name='Two pens',
        Units_Sold__c=2,
        Unit_Price__c=1.25,
        Merchandise__c = m.Id,
        Invoice_Statement__c=inv.Id);
    insert li;
} 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 line item, you can specify the line item's related merchandise record by setting the value of the `Merchandise__c` field. However, you can't change the merchandise without making a separate DML call. Similarly, when

updating a line item, if you also want to update the line item's related merchandise record, you must make two DML calls. The following example updates a line item and its related merchandise item using two `update` statements.

```
try {
    // Use a SOQL query to access data for a line item
    Line_Item__c li = [SELECT Merchandise__r.Description__c, Name
                       FROM Line_Item__c
                       WHERE Name = 'Item1' LIMIT 1];

    // Now we can change fields for both the line item and its
    // associated merchandise record
    li.Merchandise__r.Description__c = 'Hot product';
    li.Name = 'New line item';

    // To update the database, the two types of records must be
    // updated separately
    update li;          // This only updates the line item's description
    update li.Merchandise__r; // This updates the merchandise description
} catch(Exception e) {
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}
```

### 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 invoice line item with a parent invoice statement using a single `insert` statement. First, the example queries an existing merchandise item to be used for the new line item. Next, it creates a line item sObject and populates some of its fields, then creates two invoice statement objects. The first is only for the foreign key relationship, and the second is for the invoice statement creation and has more fields set. Both invoice statements 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 line item sObject. The `Database.insert` statement creates the line item with its parent invoice statement 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 Invoice_Statement called MyExtID.

```
public class ParentChildInvoiceSample {
    public static void InsertParentChildInvoice() {
        // Get an existing merchandise item
        Merchandise__c m = [SELECT Name FROM Merchandise__c LIMIT 1];
        System.assert(m != null);

        // Create an invoice line item.
        Line_Item__c invLI = new Line_Item__c(
            Merchandise__c = m.Id,
            Unit_Price__c=4,
            Units_Sold__c=1);
```

```
        // Create the parent reference.
        // Used only for foreign key reference
        // and doesn't contain any other fields.
        Invoice_Statement__c invoiceReference = new Invoice_Statement__c(
            MyExtID__c='SAP111111');
        // Set the parent reference.
        invLI.Invoice_Statement__r = invoiceReference;

        // Create the invoice statement object to insert.
        // Same as above but has additional fields.
        // Used for the insert.
        Invoice_Statement__c parentInvoice = new Invoice_Statement__c(
            Description__c='InvoiceAndStatementInsert',
            MyExtID__c='SAP111111');

        // Create the invoice and the line item.
        Database.SaveResult[] results = Database.insert(new SObject[] {
            parentInvoice, invLI });

        // 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, or the custom external ID field value, if specified.

- 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 "Creating Custom Fields" in the Salesforce Help.

### Examples

The following example updates the price for all existing merchandise items whose price is equal to 10, and also inserts a new merchandise item in a single upsert statement:

```
Merchandise__c[] mList =
            [SELECT Id, Price__c
             FROM Merchandise__c
             WHERE Price__c = 10];
for (Merchandise__c a : mList) {
    a.Price__c = 9;
}

Merchandise__c m = new Merchandise__c(
   Name='Pencil',
   Description__c='High quality pencils',
```

```
   Price__c=1.25,
   Total_Inventory__c=100);
mList.add(m);

try {
    upsert mList;
} catch (DmlException e) {
    // Process exception here
}
```

**Note:** For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 327.

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 Understanding Execution Governors and Limits). This next example uses `upsert` and an external ID field `MyExtId__c` on the Merchandise custom object. It creates two merchandise items with different external ID values, and then upserts them. If any merchandise record exists in the database with the same value for the external ID field, `upsert` updates it. Otherwise, `upsert` creates a new merchandise record.

**Note:** Before running this sample, create a custom text field on the Merchandise object named `MyExtId__c` and mark it as an external ID. For information on custom fields, see the Database.com online help.

```
public void upsertExample() {
    List<Merchandise__c> mList = new List<Merchandise__c>();

    Merchandise__c m1 = new Merchandise__c(
    Name='Erasers',
    Description__c='White erasers',
    Price__c=1.75,
    Total_Inventory__c=99,
    MyExtId__c='11111111');
    mList.add(m1);

    Merchandise__c m2 = new Merchandise__c(
    Name='Scissors',
    Description__c='Sharp scissors',
    Price__c=3,
    Total_Inventory__c=200,
    MyExtId__c='22222222');
    mList.add(m2);

    try {
        upsert mList MyExtId__c;
    } catch (DmlException e) {
        System.debug(e.getMessage());
    }
}
```

## 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 Database.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 merchandise items that are named 'Pencil':

```
Merchandise__c[] pencils = [SELECT Id, Name FROM Merchandise__c
                            WHERE Name = 'Pencil'];
try {
```

```
    delete pencils;
} catch (DmlException e) {
    // Process exception here
}
```

**Note:** For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 327.

#### 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 an invoice statement record, Apex automatically deletes any line item records associated with it. However, if a particular child record is not deletable or is currently being used, then the `delete` operation on the parent invoice statement record fails.

The `undelete` operation restores the record associations for the following types of relationships:

- All custom lookup relationships
- Tags

**Note:** Database.com only restores lookup relationships that have not been replaced.

## 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 invoice statement. The `ALL ROWS` keyword queries all rows for both top level and aggregate relationships, including deleted records and archived activities.

```
Invoice_Statement__c[] savedInvoices =
                       [SELECT Id
                        FROM Invoice_Statement__c
                        WHERE Description__c = 'My invoice' 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 327.

#### 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*

# 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.

```
Invoice_Statement__c inv = new Invoice_Statement__c();
try {
    insert inv;
} 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 |
| delete | DeleteResult Class |
| undelete | UndeleteResult 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 merchandise items, one of which doesn't have the required fields, 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 merchandise items, one of which is missing a required field
Merchandise__c[] merList = new List<Merchandise__c>{
    new Merchandise__c(Name='Pencils',
                       Description__c='Durable pencils',
                       Price__c=2,
                       Total_Inventory__c=100),
    new Merchandise__c()};
Database.SaveResult[] srList = Database.insert(merList, 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.
- The user locale for labels.
- Whether the operation allows for partial success.

The `Database.DMLOptions` class has the following properties:

- `allowFieldTruncation` 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 Database.com 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;
```

#### `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 Database.com supports, see What languages does Database.com support? in the Database.com 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.com API version 20.0 and later.

### Transaction Control

All requests are delimited by the trigger, class method, Web Service, or anonymous block that executes the Apex code. If the entire request completes successfully, all changes are 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 execution context. If you declare a savepoint as a static variable then try to use it across trigger contexts you will receive a runtime 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.

```
Invoice_Statement__c a = new Invoice_Statement__c();
insert a;
System.assertEquals(null, [SELECT Description__c FROM Invoice_Statement__c
                           WHERE Id = :a.Id].Description__c);

// Create a savepoint while the description field is null
Savepoint sp = Database.setSavepoint();

// Change the description
a.Description__c = '123';
update a;
System.assertEquals('123', [SELECT Description__c FROM Invoice_Statement__c
                            WHERE Id = :a.Id].Description__c);

// Rollback to the previous null value
Database.rollback(sp);
System.assertEquals(null, [SELECT Description__c FROM Invoice_Statement__c
                           WHERE Id = :a.Id].Description__c);
```

## 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 invoice statement 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.com API version 14.0 and earlier.
- ObjectPermissions
- PermissionSet
- PermissionSetAssignment
- QueueSObject
- SetupEntityAccess
- User

  You can insert a user in a transaction with other sObjects in Apex code saved using Salesforce.com API version 14.0 and earlier.

  You can insert a user in a transaction with other sObjects in Apex code saved using Salesforce.com 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.com API version 14.0 and earlier
- UserRole
- Custom settings in Apex code saved using Salesforce.com API version 17.0 and earlier.

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 invoice statement, which is a mixed DML operation.

```
@isTest
private class MixedDML {
    static testMethod void mixedDMLExample() {
        User u;
        Invoice_Statement__c inv;
        User thisUser = [SELECT Id FROM User WHERE Id = :UserInfo.getUserId()];
```

```
        // Insert invoice statement 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 = 'jsmtih', 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;
            inv = new Invoice_Statement__c();
            insert inv;
        }
    }
}
```

#### 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.

## sObjects That Don't Support DML Operations

Your organization contains standard objects provided by Database.com 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:

- CurrencyType
- DatedConversionRate
- Profile

## DML Operations

You can perform DML operations using the DML statements or the methods of the `Database` class.

### DML Statements

Use Data Manipulation Language (DML) operations to insert, update, merge, delete, and restore data in a database.

The following Apex DML statements are available:

***Insert Statement***
***Update Statement***
***Upsert Statement***
***Delete Statement***
***Undelete Statement***

### Insert Statement

The `insert` DML operation adds one or more sObjects 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 invoice statement:

```
Invoice_Statement__c invoice = new Invoice_Statement__c(
   Description__c = 'Invoice 1');
try {
   insert invoice;
} catch (DmlException e) {
// Process exception here
}
```

**Note:** For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 327.

### Update Statement

The `update` DML operation modifies one or more existing sObject records, such as individual invoice 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 `Description__c` field on a single invoice statement:

```
Invoice_Statement__c inv = new Invoice_Statement__c(
                              Description__c='Invoice 1');
insert inv;

Invoice_Statement__c myInvoice = [SELECT Id, Description__c
                                  FROM Invoice_Statement__c
                                  WHERE Id = :inv.Id];
myInvoice.Description__c = 'New description';

try {
    update myInvoice;
} catch (DmlException e) {
    // Process exception here
}
```

**Note:** For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 327.

### Upsert Statement

The `upsert` DML operation creates new sObject records and updates existing sObject records within a single statement, using an optional custom field to determine the presence of existing objects.

#### Syntax

```
upsert sObject opt_external_id
```

```
upsert sObject[] opt_external_id
```

***opt_external_id*** is an optional variable that specifies the custom field that should be used to match records that already exist in your organization's data. This custom field must be created with the `External Id` attribute selected. Additionally, if the field does not have the `Unique` attribute selected, 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.

If ***opt_external_id*** is not specified, the sObject record's ID field is used by default.

**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 "Creating Custom Fields" in the Database.com online help.

#### How Upsert Chooses to Insert or Update

Upsert uses the sObject record's primary key (or the external ID, if specified) to determine whether it should create a new object record or update an existing one:

- 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.

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 Database.com*.

#### Example

This example performs an upsert of a list of merchandise items.

```
List<Merchandise__c> merList = new List<Merchandise__c>();
// Fill the list with some merchandise items

try {
    upsert merList;
} catch (DmlException e) {

}
```

This next example performs an upsert of a list of merchandise items using a foreign key for matching existing records, if any.

```
List<Merchandise__c> merList = new List<Merchandise__c>();
// Fill the list with some merchandise items

try {
    // Upsert using an external ID field
    upsert merList myExtIDField__c;
} catch (DmlException e) {

}
```

### Delete Statement

The `delete` DML operation deletes one or more existing sObject records 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 merchandise items that are named 'Pencil':

```
Merchandise__c[] pencils = [SELECT Id, Name FROM Merchandise__c
                            WHERE Name = 'Pencil'];
try {
    delete pencils;
} catch (DmlException e) {
    // Process exception here
}
```

**Note:** For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 327.

## Undelete Statement

The `undelete` DML operation restores one or more existing sObject records, such as individual invoice statements. `undelete` is analogous to the UNDELETE statement in SQL.

### Syntax

```
undelete sObject | ID

undelete sObject[] | ID[]
```

### Example

The following example undeletes an invoice statement. The `ALL ROWS` keyword queries all rows for both top level and aggregate relationships, including deleted records and archived activities.

```
Invoice_Statement__c[] savedInvoices =
                       [SELECT Id
                        FROM Invoice_Statement__c
                        WHERE Description__c = 'My invoice' 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 327.

## 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 `all_or_none` 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, 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 which 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 did not 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:** During the second and third attempts, governor limits are reset to their original state before the first attempt. See Understanding Execution Governors and Limits on page 203.

## 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 Database.com, this workaround is unnecessary.

### 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 Understanding Execution Governors and Limits on page 203)

#### 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 Database.com*.

# 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 Database.com 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 merchandise items, also locks the merchandise items that are returned:

```
Merchandise__c [] merchandise = [SELECT Id FROM Merchandise__c 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, you will get a `QueryException`. Similarly, if you attempt to update a record currently locked by another client, 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.

**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 (Merchandise__c[] merchandise : [SELECT Id FROM Merchandise__c
                    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 Database.com Object Query Language (SOQL) or Database.com 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 merchandise items that are named Pen:

```
List<Merchandise__c> aa = [SELECT Id, Name FROM Merchandise__c WHERE Name = 'Pen'];
```

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 line item for the first merchandise with a total inventory greater than 1000:

```
Line_Item__c li = new Line_Item__c(
   Merchandise__c = [SELECT Name FROM Merchandise__c
   WHERE Total_Inventory__c > 1000 LIMIT 1].Id);
li.Name='Two items';
li.Invoice_Statement__c=invoiceID;
```

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 merchandise items with a total inventory greater than 1000:

```
Integer i = [SELECT COUNT() FROM Merchandise__c WHERE Total_Inventory__c > 1000];
```

You can also operate on the results using standard arithmetic:

```
Integer j = 5 * [SELECT COUNT() FROM Merchandise__c];
```

For a full description of SOQL query syntax, see the *Database.com 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 merchandise items, inventory statements, and line items that have fields that begin with the phrase map:

```
List<List<SObject>> searchList = [FIND 'map*' IN ALL FIELDS RETURNING
                                  Merchandise__c (Id, Name), Invoice_Statement__c,
Line_Item__c];
```

**Note:**

The syntax of the `FIND` clause in Apex differs from the syntax of the `FIND` clause in the SOAP API:

- In Apex, the value of the `FIND` clause is demarcated with single quotes. For example:

  ```
  FIND 'map*' IN ALL FIELDS RETURNING Merchandise__c (Id, Name), Invoice_Statement__c,
   Line_Item__c
  ```

- In the Force.com API, the value of the `FIND` clause is demarcated with braces. For example:

  ```
  FIND {map*} IN ALL FIELDS RETURNING Merchandise__c (Id, Name), Invoice_Statement__c,
   Line_Item__c
  ```

From `searchList`, you can create arrays for each object returned:

```
Merchandise__c [] merch = ((List<Merchandise__c>)searchList[0]);
Invoice_Statement__c [] invoices = ((List<Invoice_Statement__c>)searchList[1]);
Line_Item__c [] li = ((List<Line_Item__c>)searchList[2]);
```

For a full description of SOSL query syntax, see the *Database.com 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 Invoice_Statement__c(Description__c = 'Singha');
Invoice_Statement__c inv = [SELECT Id FROM Invoice_Statement__c
                            WHERE Description__c = 'Singha' LIMIT 1];
// Note that description is not queried
String s = [SELECT Id FROM Invoice_Statement__c
               WHERE Description__c = 'Singha' LIMIT 1].Description__c;
```

The following is the same code example rewritten so it does not produce a runtime error. Note that `Description__c` has been added as part of the select statement, after `Id`.

```
insert new Invoice_Statement__c(Description__c = 'Singha');
Invoice_Statement__c inv = [SELECT Id FROM Invoice_Statement__c
                            WHERE Description__c = 'Singha' LIMIT 1];
// Note that description is now queried
String s = [SELECT Id,Description__c FROM Invoice_Statement__c
               WHERE Description__c = 'Singha' LIMIT 1].Description__c;
```

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 merchandise record in the list, and then dereferences the record's `Price__c` field:

```
Decimal price = [SELECT Price__c FROM Merchandise__c
                 WHERE Name = 'Pen'][0].Price__c;

// When only one result is returned in a SOQL query, it is not necessary
// to include the list's index.
Decimal price = [SELECT Price__c FROM Merchandise__c
                 WHERE Name = 'Pen' LIMIT 1].Price__c;
```

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 Merchandise__c];
```

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 Line_Item__c sObject has both an `Invoice_Statement__c` field of type ID, and an `Invoice_Statement__r` field that points to the associated sObject record itself.

The ID field can be used to change the invoice statement with which the line item is associated, while the sObject reference field can be used to access data from the invoice statement. 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 invoice statement and a line item can be associated with one another, and then how the line item can be used to modify a field on the invoice statement:

**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 324 and Update Statement on page 324.

```
// Create a merchandise item to be set for the line item
Merchandise__c m = new Merchandise__c(
   Name='Pencils',
   Description__c='Durable pencils',
   Price__c=1.25,
   Total_Inventory__c=100);
// Inserting the record automatically assigns a
// value to its ID field.
insert m;

// Create an invoice statement
Invoice_Statement__c inv = new Invoice_Statement__c(
                      Description__c = 'Invoice 1');
insert inv;

// Create a new line item and associate it with
// the invoice statement and merchandise item
```

```
// through their respective IDs.
Line_Item__c li = new Line_Item__c(
                    Name='Two pencils',
                    Units_Sold__c=2,
                    Unit_Price__c=5,
                    Merchandise__c = m.Id,
                    Invoice_Statement__c=inv.Id);
insert li;

// A SOQL query accesses data for the inserted line item,
// including a populated Invoice_Statement__r field
li = [SELECT Invoice_Statement__r.Description__c
      FROM Line_Item__c WHERE Id = :li.Id];

// Now fields in both records can be changed through the
// returned line item object
li.Invoice_Statement__r.Description__c = 'Updated description';
li.Units_Sold__c = 3;

// To update the database, the two types of records must be
// updated separately
update li;          // This only changes the line item's units sold
update li.Invoice_Statement__r; // This updates the invoice's description
```

**Note:** The expression `li.Invoice_Statement__r.Description__c`, 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 `li.Invoice_Statement__r` is null, then `li.Invoice_Statement__r.Description__c` 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 `li.Invoice_Statement__r` is null, then `li.Invoice_Statement__r.Description__c` *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:

```
Invoice_Statement__c refInvoice = new Invoice_Statement__c(externalId__c = '12345')
Merchandise__c refMerch = new Merchandise__c(externalId__c = '12345', ...)

Line_Item__c li = new Line_Item__c(
                        Name='Two pencils',
                        Units_Sold__c=2,
                        Unit_Price__c=5,
                        Merchandise__c = refMerch,
                        Invoice_Statement__c=refInvoice);
```

This inserts a new line item with the invoice statement ID equal to the invoice statement with the `external_id` equal to '12345'. If there is no such invoice statement, the insert fails. The same is true also for the merchandise ID.

**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 Understanding Execution Governors and Limits on page 203.

```
Invoice_Statement__c refInvoice = [SELECT Id FROM Invoice_Statement__c WHERE
externalId__c='12345'];
Merchandise__c refMerch = [SELECT Id FROM Merchandise__c WHERE externalId__c='12345'];

Line_Item__c li = new Line_Item__c(
                        Name='Two pencils',
```

```
                            Units_Sold__c=2,
                            Unit_Price__c=5,
                            Merchandise__c = refMerch.Id,
                            Invoice_Statement__c=refInvoice.Id);
insert li;
```

## 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 Merchandise__r.Name FROM Line_Item__c
              WHERE Name = 'Two pencils'].Merchandise__r.Name);
```

Additionally, parent-child relationships in sObjects act as SOQL queries as well. For example:

```
for (Invoice_Statement__c inv : [SELECT Id, Description__c,
                                 (SELECT Name FROM Line_Items__r)
                                 FROM Invoice_Statement__c
                                 WHERE Description__c = 'Invoice 1']) {
     Line_Item__c[] lis = inv.Line_Items__r;
     system.debug('lis.size(): ' + lis.size());
}
```

## 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 *Database.com 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 *Database.com 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 may return 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 through the use of internal calls to `query` and `queryMore`.

For example, if the results are too large, the syntax below causes a runtime exception:

```
Merchandise__c[] merchandise = [SELECT Id FROM Merchandise__c];
```

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 (Merchandise__c m : [SELECT Id, Name FROM Merchandise__c
                  WHERE Name LIKE 'p%']) {
    // Your code without DML statements here
}

// Use this format for efficiency if you are executing DML statements
// within the for loop
for (List<Merchandise__c> ml : [SELECT Id, Name FROM Merchandise__c
                           WHERE Name LIKE 'p%']) {
    // Your code here
    update ml;
}
```

The following example demonstrates a SOQL query `for` loop used to mass update records. Suppose you want to increase the price of a merchandise item by 10% across all records for merchandise items whose names includes the word 'pen':

```
public void massUpdate() {
    for (List<Merchandise__c> merchList : [SELECT Name FROM Merchandise__c]) {
        for(Merchandise__c m : merchList) {
            if (m.Name.contains('pen')) {
                m.Price__c *= 1.1;
            }
        }
        update merchList;
    }
}
```

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 122.

### More Efficient SOQL Queries

For best performance, SOQL queries must be selective, particularly for queries inside of triggers. To avoid long execution times, non-selective SOQL queries may be terminated by the system. Developers will 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 records for the first million records and less than 5% of the records after the first million records, up to a maximum of 333,000 records. In some circumstances, for example with a query filter that is an indexed standard field, the threshold may 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), and custom fields marked as External ID or Unique.
- Fields that aren't indexed by default might later be automatically indexed if the Database.com optimizer recognizes that an index will improve performance for frequently run queries.
- Salesforce.com 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.) Note that new data types, typically complex ones, may be added to Database.com and fields of these types may not allow custom indexing.
- Typically, a custom index won't be used in these cases:
  - ◊ The value(s) queried for exceeds the system-defined threshold mentioned above
  - ◊ 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 333,000. This is because the `CONTAINS` operator requires a full scan of the index. Note that this threshold is subject to change.
  - ◊ When comparing with an empty value (`Name != ''`)

  However, there are other complex scenarios in which custom indexes won't be used. Contact your salesforce.com 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, we will assume there are more than 100,000 records (including soft-deleted records, that is, deleted records that are still in the Recycle Bin) for the Merchandise__c sObject.

Query 1:

```
SELECT Id FROM Merchandise__c WHERE Id IN (<list of merchandise IDs>)
```

The `WHERE` clause is on an indexed field (Id). If `SELECT COUNT() FROM Merchandise__c WHERE Id IN (<list of merchandise IDs>)` returns fewer records than the selectivity threshold, the index on Id is used. This will typically be the case since the list of IDs only contains a small amount of records.

Query 2:

```
SELECT Id FROM Merchandise__c WHERE Name != ''
```

Since Merchandise__c 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 Merchandise__c 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 Merchandise__c 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<Merchandise__c> merchandiseItems = [SELECT Id FROM Merchandise__c];

// 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.
Merchandise__c merch = [SELECT Id FROM Merchandise__c];
String name = [SELECT Name FROM Merchandise__c].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 Owner relationship field of a Merchandise__c custom object could be a User or a Group.

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 Database.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 Merchandise__c records that are related to a Group or User via the Owner field.

```
List<Merchandise__c> = [SELECT Name FROM Merchandise__c WHERE Owner.Type IN ('User',
'Group')];
```

Another approach would be to use the `TYPEOF` clause in the SOQL `SELECT` statement. This example also queries Merchandise__c records that are related to a User or Group via the Owner field.

```
List<Merchandise__c> = [SELECT TYPEOF Owner WHEN User THEN LastName WHEN Group THEN Email
END FROM Merchandise__c];
```

**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.com.

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 a User or Group is related to a Merrchandise__c.

```
Merchandise__c myMerch = merchFromQuery;
if (myMerch.Owner instanceof User) {
    // myMerch.Owner references a User, so process accordingly
} else if (myMerch.Owner instanceof Group) {
    // myMerch.Owner references a Group, 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 are 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:

```
Merchandise__c A = new Merchandise__c(
                    Name='Pen',
                    Description__c='Black pens',
                    Price__c=1.25,
                    Total_Inventory__c=100);
insert A;
Merchandise__c B;

// A simple bind
B = [SELECT Id FROM Merchandise__c WHERE Id = :A.Id];

// A bind with arithmetic
B = [SELECT Id FROM Merchandise__c
     WHERE Name = :('x' + 'xx')];

String s = 'XXX';

// A bind with expressions
B = [SELECT Id FROM Merchandise__c
     WHERE Name = :'XXXX'.substring(0,3)];

// A bind with an expression that is itself a query result
B = [SELECT Id FROM Merchandise__c
     WHERE Name = :[SELECT Name FROM Merchandise__c
                    WHERE Id = :A.Id].Name];

Line_Item__c C = new Line_Item__c(
   Name='Two pens',
   Units_Sold__c=2,
```

```
   Unit_Price__c=1.25,
   Merchandise__c = m.Id,
   Invoice_Statement__c=inv.Id));

insert new Line_Item__c[]{C,
                          new Line_Item__c(Name='Five pens',
                             Units_Sold__c=5,
                             Unit_Price__c=1.25,
                             Merchandise__c = m.Id,
                             Invoice_Statement__c=inv.Id)};

// Binds in both the parent and aggregate queries
B = [SELECT Id, (SELECT Id FROM Line_Item__c
                 WHERE Id = :C.Id)
     FROM Merchandise__c
     WHERE Id = :A.Id];

// One line item returned
SObject D = B.getSObjects('Line_Items__r');
Line_Item__c li = (Line_Item__c)D;

// A limit bind
Integer i = 1;
B = [SELECT Id FROM Merchandise__c LIMIT :i];

// An OFFSET bind
Integer offsetVal = 10;
List<Merchandise__c> offsetList = [SELECT Id FROM Merchandise__c 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
Invoice_Statement__c[] cc = [SELECT Id FROM Invoice_Statement__c LIMIT 2];
Line_Item__c[] tt = [SELECT Id,Name FROM Line_Item__c WHERE Invoice_Statement__c IN :cc];

// An IN-bind with a String list
String[] ss = new String[]{'a0290000000UuT7', 'a0290000000UuSn'};
Merchandise__c[] aa = [SELECT Id FROM Merchandise__c
                WHERE Id 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
                                     Merchandise__c (Id, Name WHERE Name LIKE :myString2
                                              LIMIT :myInt3),
                                     Invoice_Statement__c,
                                     Line_Item__c,
                                  WITH DIVISION =:myString4
                                  LIMIT :myInt5];
```

## 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 For example:

```
System.assertEquals(2, [SELECT COUNT() FROM Merchandise__c WHERE Name LIKE 'p%' 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 = 'Pen';
for (Merchandise__c a : [SELECT Id, Name from Merchandise__c
                  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 merchandise records from a SOQL query
List<Merchandise__c> merch = [SELECT Id, Name
                              FROM Merchandise__c
                              WHERE Name = 'Pen'];

// Loop through the list and update the Name field
for(Merchandise__c m : merch){
   m.Name = 'Pencil';
}

// Update the database
update merch;
```

### 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 runtime exception occurs if you use a query containing an aggregate function that returns more than 2000 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 Invoice_Statement__c[]{
                    new Invoice_Statement__c(Description__c = 'yyy'),
                    new Invoice_Statement__c(Description__c = 'yyy'),
                    new Invoice_Statement__c(Description__c = 'yyy')};

// The single sObject format executes the for loop once per returned record
Integer i = 0;
for (Invoice_Statement__c tmp : [SELECT Id FROM Invoice_Statement__c
                                 WHERE Description__c = 'yyy']) {
    i++;
}
System.assert(i == 3); // Since there were three invoices 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 (Invoice_Statement__c[] tmp : [SELECT Id FROM Invoice_Statement__c
                                   WHERE Description__c = 'yyy']) {
    j = tmp.size();
    i++;
}
System.assert(j == 3); // The list should have contained the three invoices
                       // named 'yyy'
System.assert(i == 1); // Since a single batch can hold up to 100 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 Understanding Execution Governors and Limits on page 203.
- 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 of a retrieved sObject inside the loop, or when getting the size of such a record set. To avoid getting this exception, use a `for` loop to iterate over the child records, as follows.

```
Integer count=0;
for (Line_Item__c li : returnedInvoice.Line_Items__r) {
    count++;
    // Do some other processing
}
```

# 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 invoice statements
List<Invoice_Statement__c> invoices = new List<Invoice_Statement__c>();
```

### 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 invoice statements to the return value of a SOQL query. The query returns up to 1,000 returns records containing the Id and Description fields.

```
// Create a list of invoice statement records from a SOQL query
List<Invoice_Statement__c> invoicesFromSOQL =
        [SELECT Id, Description__c FROM Invoice_Statement__c 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<Invoice_Statement__c> myList = new List<Invoice_Statement__c>(); // Define a new list
// Create the invoice first
Invoice_Statement__c inv = new Invoice_Statement__c(Description__c='New invoice');
myList.add(inv);                    // Add the invoice statement sObject
Invoice_Statement__c inv2 = 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 invoice statements.

```
// Define the list
List<Invoice_Statement__c> invList = new List<Invoice_Statement__c>();
// Create invoice statement sObjects
```

```
Invoice_Statement__c inv1 = new Invoice_Statement__c(Description__c='Invoice 1');
Invoice_Statement__c inv2 = new Invoice_Statement__c(Description__c='Invoice 2');
// Add invoice statements to the list
invList.add(inv1);
invList.add(inv2);
// Bulk insert the list
insert invList;
```

### 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
   Invoice_Statement__c a = new Invoice_Statement__c();
   List<Invoice_Statement__c> invs = new List<Invoice_Statement__c>{a, a};

   // Attempt to insert it
   insert invs;

   // 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 invoice statements using the array notation.

```
Invoice_Statement__c[] invoices = new Invoice_Statement__c[1];
```

And this example adds an element to the list using square brackets.

```
invoices[0] = new Invoice_Statement__c(Description__c='New invoice');
```

These are some additional examples of using the array notation with sObject lists.

| Example | Description |
|---|---|
| `List<Invoice_Statement__c> accts = new Invoice_Statement__c[]{};` | Defines an empty list that can hold Invoice_Statement__c objects. |
| `List<Invoice_Statement__c> invs = new Invoice_Statement__c[] {new Invoice_Statement__c(), null, new Invoice_Statement__c()};` | Defines a list that can hold Invoice_Statement__c objects and allocates memory for three invoice statements, including a new Invoice_Statement__c object in the first position, `null` in the second position, and another new Invoice_Statement__c object in the third position. |
| `List<Invoice_Statement__c> invs = new List<Invoice_Statement__c>(otherList);` | Defines a list of Invoice_Statement__c objects 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.
2. The Name field, if applicable.
3. Standard fields, starting with the fields that come first in alphabetical order, except for the Id and Name fields.
4. Custom fields, starting with the fields that come first in alphabetical order.

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 Merchandise__c custom objects. 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 the Merchandise__c custom object. The implementation of the `compareTo` method in this class compares two merchandise objects based on the Price field—the class member variable contained in this instance, and the merchandise object passed into the method.

```
global class MerchandiseWrapper implements Comparable {

    public Merchandise__c merchItem;

    // Constructor
    public MerchandiseWrapper(Merchandise__c m) {
        merchItem = m;
    }

    // Compare merchandise items based on the merchandise price.
    global Integer compareTo(Object compareTo) {
        // Cast argument to MerchandiseWrapper
        MerchandiseWrapper compareToMerch = (MerchandiseWrapper)compareTo;

        // The return value of 0 indicates that both elements are equal.
        Integer returnValue = 0;
        if (merchItem.Price__c > compareToMerch.merchItem.Price__c) {
            // Set return value to a positive value.
            returnValue = 1;
        } else if (merchItem.Price__c < compareToMerch.merchItem.Price__c) {
            // Set return value to a negative value.
            returnValue = -1;
        }

        return returnValue;
    }
}
```

This example provides a test for the `MerchandiseWrapper` class. It sorts a list of `MerchandiseWrapper` objects and verifies that the list elements are sorted by the merchandise price.

```
@isTest
private class MerchandiseWrapperTest {
    static testmethod void test1() {
        MerchandiseWrapper[] merchList = new List<MerchandiseWrapper>();
        merchList.add( new MerchandiseWrapper(new Merchandise__c(
            Name='Pens',
            Description__c='Red pens',
            Price__c=2,
            Total_Inventory__c=1000)));
        merchList.add( new MerchandiseWrapper(new Merchandise__c(
            Name='Notebooks',
            Description__c='Cool notebooks',
            Price__c=3.50,
            Total_Inventory__c=2000)));
        merchList.add( new MerchandiseWrapper(new Merchandise__c(
            Name='Pens',
            Description__c='Blue pens',
            Price__c=1.75,
            Total_Inventory__c=800)));

        // Sort the wrapper objects using the implementation of the
        // compareTo method.
        merchList.sort();

        // Verify the sort order
        System.assertEquals('Pens', merchList[0].merchItem.Name);
        System.assertEquals(1.75, merchList[0].merchItem.Price__c);
        System.assertEquals('Pens', merchList[1].merchItem.Name);
        System.assertEquals(2, merchList[1].merchItem.Price__c);
        System.assertEquals('Notebooks', merchList[2].merchItem.Name);
        System.assertEquals(3.5, merchList[2].merchItem.Price__c);
```

```
        // Write the sorted list contents to the debug log.
        System.debug(merchList);
    }
}
```

## 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 Invoice_Statement__c name is assigned to `descriptionLength`.

```
Integer descriptionLength = new Invoice_Statement__c []{
   new Invoice_Statement__c (Description__c='My invoice')}[0].Description__c.length();
```

In the above, `new Invoice_Statement__c[]` generates a list.

The list is populated with one element by the `new` statement `{new Invoice_Statement__c(Description__c='My invoice')}`.

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 `Description__c.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 descChange = [SELECT Description__c
                     FROM Invoice_Statement__c][0].Description__c.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 invoice statements with the same name to a set, with no other fields set, only one sObject is added to the set.

```
// Create two invoice statements, a1 and a2
Invoice_Statement__c a1 = new Invoice_Statement__c(Description__c='desc');
Invoice_Statement__c a2 = new Invoice_Statement__c(Description__c='desc');

// Add both invoices to the new set
Set<Invoice_Statement__c> mySet =
              new Set<Invoice_Statement__c>{a1, a2};

// Verify that the set only contains one item
System.assertEquals(mySet.size(), 1);
```

If you add a description to one of the invoice statements, it is considered unique and both invoices are added to the set.

```
// Create two invoice statements, a1 and a2.
// Add a description to a2.
Invoice_Statement__c a1 = new Invoice_Statement__c();
Invoice_Statement__c a2 = new Invoice_Statement__c(Description__c='desc');

// Add both invoices to the new set
Set<Invoice_Statement__c> mySet =
```

```
            new Set<Invoice_Statement__c>{a1, a2};

// Verify that the set only contains one item
System.assertEquals(mySet.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 Merchandise__c.

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 a merchandise item (a specific sObject type). This example shows how to define a map whose keys are of type ID and whose values are of type Merchandise__c.

```
Map<ID, Merchandise__c> m = new Map<ID, Merchandise__c>();
```

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 merchandise lists and adds one entry using the merchandise list created earlier.

```
// Merchandise__c[] is synonymous with List<Merchandise__c>
Merchandise__c[] merchList = new Merchandise__c[5];
Map<Integer, List<Merchandise__c>> m4 = new Map<
                    Integer, List<Merchandise__c>>{1 => merchList};
```

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 merchandise items with their `Id` and `Name` fields. The `new` operator uses the returned list of items to create a map.

```
// Populate map from SOQL query
Map<ID, Merchandise__c> m = new Map<ID, Merchandise__c>([SELECT Id, Name FROM Merchandise__c
 LIMIT 10]);
// After populating the map, iterate through the map entries
for (ID idKey : m.keyset()) {
    Merchandise__c 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 a Merchandise__c sObject as the value.

```
// Define a new merchandise item
Merchandise__c mer = new Merchandise__c();
```

```
// Define a new map
Map<Integer, Merchandise__c> m = new Map<Integer, Merchandise__c>();
// Insert a new key-value pair in the map
m.put(1, mer);
// Assert that the map contains a key
System.assert(!m.containsKey(3));
// Retrieve a value, given a particular key
Merchandise__c a = m.get(1);
// Return a set that contains all of the keys in the map
Set<Integer> s = m.keySet();
```

### 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 invoice and add it to the map
Invoice_Statement__c inv1 = new Invoice_Statement__c(Name='Inv1');
Map<sObject, Integer> m = new Map<sObject, Integer>{
    inv1 => 1};

// Get inv1's value from the map.
// Returns the value of 1.
System.assertEquals(1, m.get(inv1));
// Id field is null.
System.assertEquals(null, inv1.Id);

// Insert inv1.
// This causes the ID field on inv1 to be auto-filled
insert inv1;
// Id field is now populated.
System.assertNotEquals(null, inv1.Id);

// Get inv1'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 inv1 was originally added to the map
// before the insert operation, the ID of inv1 was null.
System.assertEquals(null, m.get(inv1));
```

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.

- 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.

## 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 Invoice_Statement__c 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 invoice statement as the generic type sObject
sObject s = new Invoice_Statement__c();

// Verify that the generic sObject is an Invoice_Statement__c sObject
System.assert(s.getsObjectType() == Invoice_Statement__c.sObjectType);

// Get the sObject describe result for the
// invoice statement object
Schema.DescribeSObjectResult r =
         Invoice_Statement__c.sObjectType.getDescribe();

// Get the field describe result for the Status__c
// field on the Invoice_Statement__c object
Schema.DescribeFieldResult f =
         Schema.sObjectType.Invoice_Statement__c.fields.Status__c;

// Verify that the field token is the token for the
// Status__c field on an Invoice_Statement__c object
System.assert(f.getSObjectField() == Invoice_Statement__c.Status__c);

// Get the field describe result from the token
f = f.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 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 Invoice_Statement__c.
- 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 Invoice_Statement__c sObject is returned:

```
Schema.sObjectType t = Invoice_Statement__c.sObjectType;
```

The following also returns a token for the Invoice_Statement__c sObject:

```
Invoice_Statement__c A = new Invoice_Statement__c();
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:

```
public class sObjectTest {
{
// Create a generic sObject variable s
SObject s = Database.query('SELECT Id FROM Invoice_Statement__c LIMIT 1');

// Verify if that sObject variable is an Invoice_Statement__c token
System.assertEquals(s.getSObjectType(), Invoice_Statement__c.sObjectType);

// Create a list of generic sObjects
List<sObject> l = new Invoice_Statement__c[]{};

// Verify if the list of sObjects contains Invoice_Statement__c tokens
System.assertEquals(l.getSObjectType(), Invoice_Statement__c.sObjectType);
}
}
```

### Obtaining sObject Describe Results Using Tokens

To access the describe result for an sObject, call the `getDescribe` method on an sObject token

`Schema.DescribeSObjectResult` is the data type for an sObject describe result.

The following example uses the `getDescribe` method on an sObject token:

```
Schema.DescribeSObjectResult D = Invoice_Statement__c.sObjectType.getDescribe();
```

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, `Invoice_Statement__c.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 Invoice_Statement__c object's `Status__c` field:

```
Schema.SObjectField F = Invoice_Statement__c.Status__c;
```

In the following example, the field token is returned from the field describe result:

```
// Get the describe result for the status field on the
// Invoice_Statement__c object
Schema.DescribeFieldResult f =
       Schema.sObjectType.Invoice_Statement__c.fields.Status__c;

// Verify that the field token is the token for
// the status field on an Invoice_Statement__c object
System.assert(f.getSObjectField() == Invoice_Statement__c.Status__c);

// Get the describe result from the token
f = f.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 F = Invoice_Statement__c.Status__c.getDescribe();
```

This example uses the `fields` member variable method:

```
Schema.DescribeFieldResult F =
           Schema.SObjectType.Invoice_Statement__c.fields.Status__c;
```

In the example above, the system uses special parsing to validate that the final member variable (`Status__c`) 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 (`Invoice_Statement__c`) and validates that the field name following the `fields` member variable is legitimate. The `fields` member variable only works when used in this manner.

You can only have 100 `fields` member variable statements in an Apex class or trigger.

**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> M =
          Schema.SObjectType.Invoice_Statement__c.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.

**Note:** A field describe 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 not within an installed managed package.

## Understanding Describe Information Permissions

Apex generally runs in system mode. All classes and triggers that 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.

## 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. Using this method, you can describe up to 100 sObjects.

This example gets describe metadata information for two sObject types—The Invoice_Statement__c and the Merchandise__c custom objects. 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[]{'Invoice_Statement__c','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.');
    }
}
```

## 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.com 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.

## 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> L = 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 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 Understanding Execution Governors and Limits on page 203.

For a full description of SOQL query syntax, see Salesforce Object Query Language (SOQL) in the *Database.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> L = 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> L = 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> L = 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 runtime 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 runtime, 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
Merchandise__c(id,name),Invoice_Statement__c';
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 Merchandise__c are first, then Invoice_Statement__c.

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 Understanding Execution Governors and Limits on page 203.

For a full description of SOSL query syntax, see Salesforce Object Search Language (SOSL) in the *Database.com SOQL and SOSL Reference*.

### 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. This 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.

## 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 Invoice_Statement__c). For example:

```
// Get a new invoice statement
Invoice_Statement__c A = new Invoice_Statement__c();
// Get the token for the invoice statement
Schema.sObjectType tokenA = A.getSObjectType();
// The following produces an error because the token
// is a generic sObject, not an Invoice_Statement__c
// Invoice_Statement__c B = tokenA.newSObject();
// The following works because the token is cast back
// into an Invoice_Statement__c
Invoice_Statement__c B = (Invoice_Statement__c)tokenA.newSObject();
```

Though the sObject token `tokenA` is a token of Invoice_Statement__c, it is considered an sObject because it is accessed separately. It must be cast back into the concrete sObject type Invoice_Statement__c to use the `newSObject` method. For more information on casting, see Classes and Casting on page 77.

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 Invoice_Statement__c LIMIT 1')[0].
               getSObjectType().newSObject([SELECT Id
               FROM Invoice_Statement__c 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 invoice statement.

```
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 = 'Invoice_Statement__c';

        // Create a new sObject by passing the sObject type as an argument.
        Invoice_Statement__c inv =
(Invoice_Statement__c)DynamicSObjectCreation.createObject(typeName);
        // Verify that the sObject type name of the object created ends
        //   with the requested type since it can contain a namespace prefix.
        System.assert(String.valueOf(inv.getSobjectType()).endsWith(typeName));

        // Set fields for the sObject.
        inv.Description__c = 'Invoice 1';
        insert inv;

        // Verify the new sObject got inserted.
        Invoice_Statement__c[] invList = [SELECT Description__c from Invoice_Statement__c
                                          WHERE Id = :inv.Id];
        system.assert(invList.size() == 1);
    }
}
```

### 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 `Status__c` is used:

```
SObject s = [SELECT Status__c FROM Invoice_Statement__c LIMIT 1];
Object o = s.get('Status__c');
s.put('Status__c', 'abc');
```

The following example uses the `Status__c` field's token instead:

```
Schema.DescribeFieldResult f = Schema.sObjectType.Invoice_Statement__c.fields.Status__c;
Sobject s = Database.query('SELECT Status__c FROM Invoice_Statement__c LIMIT 1');
s.put(f.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, Value__c, Merchandise__r.Name FROM Line_Item__c LIMIT 1');
SObject a = c.getSObject('Merchandise__r');
```

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, Description__c, ' +
            '(SELECT Value__c FROM Line_Items__r LIMIT 1) ' +
             'FROM Invoice_Statement__c';
SObject[] queryParentObject = Database.query(queryString);

for (SObject parentRecord : queryParentObject){
    Object ParentFieldValue = parentRecord.get('Description__c');
    // Prevent a null relationship from being accessed
    SObject[] childRecordsFromParent =
          parentRecord.getSObjects('Line_Items__r');
    if (childRecordsFromParent != null) {
        for (SObject childRecord : childRecordsFromParent){
            Object ChildFieldValue1 = childRecord.get('Value__c');
            System.debug('Invoice Description: ' + ParentFieldValue +
            '. Line Item Value: '+ ChildFieldValue1);
        }
    }
}
```

# 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 156.

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.com 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.
  public static void m() {   }

}

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.
  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 exends 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.

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 156.

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 merchandise's price field before updating it:

```
if (Schema.sObjectType.Merchandise__c.fields.Price__c.isUpdateable()) {
   // Update merchandise price
}
```

To check the field-level create permission of the merchandise's price field before creating a new merchandise item:

```
if (Schema.sObjectType.Merchandise__c.fields.Price__c.isCreateable()) {
   // Create new merchandise
}
```

To check the field-level read permission of the merchandise's price field before querying for this field:

```
if (Schema.sObjectType.Merchandise__c.fields.Price__c.isAccessible()) {
   Merchandise__c merch = [SELECT Price__c FROM Merchandise__c WHERE Id= :Id];
}
```

To check the object-level permission for the merchandise object before deleting a merchandise item.

```
if (Schema.sObjectType.Merchandise__c.isDeletable()) {
   // Delete merchandise
}
```

Sharing rules are distinct from object-level and field-level permissions. They can coexist. If sharing rules are defined in Database.com, 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 Database.com 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 Database.com online help.

## Understanding Sharing

*Sharing* enables record-level access control for all custom objects. 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

Database.com 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. 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 Database.com 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 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 Database.com 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 Database.com 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) |
|---|---|
| Associated record owner or sharing | `ImplicitParent` |
| Owner | `Owner` |
| Sharing Rule | `Rule` |

- User Managed Sharing

| `Reason` Field Value | `rowCause` Value (Used in Apex or the Force.com API) |
|---|---|
| Manual Sharing | `Manual` |

- 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. |
| 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. |
| Full Access | All | The specified user or group can view, edit, transfer, share, and delete the record.<br>  **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 custom object for which you want to share. 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" in the Database.com 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 . Valid values are:<br>• Edit<br>• Read<br>• All<br><br>**Note:** The All access level can only be used by Force.com managed sharing.<br><br>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 144. |
| 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, 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 Database.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 144.

### 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 926.

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.

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 144.

## Recalculating Apex Managed Sharing

Database.com 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 Database.com-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 Database.com-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 179.

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 144.

### 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**.

## 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, 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 Database.com 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 |

Apex can access both custom setting types—list and hierarchy.

## 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*

# WAYS TO INVOKE APEX

# 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

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 and then uses an anonymous block of Apex to insert a new record using the given input.

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 Database.com 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 269)
- 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.

# 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
- 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.

Triggers can be divided into two types:

- *Before* triggers can be used to update or validate record values before they are saved to the database.
- *After* triggers can be used to access field values that are set by the database (such as a record's `Id` or `lastUpdated` field), and to affect changes in other records, such as logging into an audit table or firing asynchronous events with a queue.

Triggers can also modify other records of the same type as the records that initially fired the trigger. For example, suppose you created a merchandise object, if a trigger fires after an update of merchandise record `A`, the trigger can also modify merchandise record `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 Understanding Execution Governors and Limits on page 203.

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.

## Implementation Considerations

Before creating triggers, consider the following:

- `upsert` triggers fire both before and after `insert` or before and after `update` triggers as appropriate.
- Triggers that execute after a record has been undeleted only work with specific objects. See Triggers and Recovered Records on page 165.
- 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.
- For Apex saved using Salesforce.com API version 20.0 or earlier, if an API call causes a trigger to fire, the chunk of 200 records to process is further split into chunks of 100 records. For Apex saved using Salesforce.com API version 21.0 and later, no further splits of API chunks occur. Note that static variable values are reset between API batches, but governor limits are not. Do not use static variables to track state information between API batches.

# 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:**

- You can only use the `webService` keyword in a trigger when it is in a method defined as asynchronous; that is, when the method is defined with the `@future` keyword.
- 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 Invoice_Statement__c object:

```
trigger myInvoiceTrigger on Invoice_Statement__c (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 runtime 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 Web service or an `executeanonymous()` call. |
| `isInsert` | Returns true if this trigger was fired due to an insert operation. |
| `isUpdate` | Returns true if this trigger was fired due to an update operation. |
| `isDelete` | Returns true if this trigger was fired due to a delete operation. |
| `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 Apex or the API.) |
| `new` | Returns a list of the new versions of the sObject records.<br>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.<br>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.<br>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.<br>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 t on Invoice_Statement__c (after insert) {
    for (Invoice_Statement__c a : Trigger.new) {
        // Iterate over each sObject
    }

    // This single query finds every line item that is
    // associated with any of the triggering invoice statements.
    // 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.
    Line_Item__c[] li = [SELECT Name FROM Line_Item__c
                         WHERE Invoice_Statement__r.Id 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 myInvoiceTrigger on Invoice_Statement__c(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 (Invoice_Statement__c a : Trigger.old) {
            if (a.Description__c != '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 (Invoice_Statement__c a : Trigger.new) {
            if (a.Description__c == 'bad') {
                a.name.addError('Invalid invoice');
            }
    }
    if (Trigger.isInsert) {
        for (Invoice_Statement__c a : Trigger.new) {
            System.assertEquals('some description', a.Description__c);
            System.assertEquals('Open', a.Status__c);
        }

// If the trigger is not a before trigger, it must be an after trigger.
} else {
    if (Trigger.isInsert) {
        List<Line_Item__c> li = new List<Line_Item__c>();
        Merchandise__c m = new Merchandise__c(
            Name='Pencils',
            Description__c='Durable pencils',
            Price__c=5,
            Total_Inventory__c=100);
        insert m;
        for (Invoice_Statement__c a : Trigger.new) {
            if(a.Description__c == 'Invoice A') {
                li.add(new Line_Item__c(Name='Some pencils',
                    Units_Sold__c =2,
                    Unit_Price__c=5,
                    Invoice_Statement__c = a.Id,
                    Merchandise__c = m.Id));
            }
        }
      insert li;
    }
  }
}}}
```

## 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 Database.com:

1. For a custom object, click **Create** > **Objects** and click the name of the object.

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.
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:**

   - You can only use the `webService` keyword in a trigger when it is in a method defined as asynchronous; that is, when the method is defined with the `@future` keyword.
   - 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.

### The Apex Trigger Editor

When editing Apex, 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 through the `undelete` DML statement. These are also called undeleted records.

The `after undelete` trigger events only run on top-level objects.

## Triggers and Order of Execution

When you save a record with an `insert`, `update`, or `upsert` statement, Database.com performs the following events in order.

**Note:** Before Database.com 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, Database.com:

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.Database.com 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.
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 Database.com doesn't run a second time (when the request comes from a standard UI edit page) is the enforcement of layout-specific rules.
5. Saves the record to the database, but doesn't commit yet.
6. Executes all `after` triggers.
7. Executes assignment rules.
8. Executes auto-response rules.
9. Executes workflow rules.
10. If there are workflow field updates, updates the record again.
11. 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 are not run again.
12. 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.
13. If the parent record is updated, and a grand-parent 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. Grand-parent record goes through save procedure.
14. Executes Criteria Based Sharing evaluation.
15. Commits all DML operations to the database.
16. Executes post-commit logic, such as sending email.

**Note:** During a recursive save, Database.com skips steps 7 through 13.

### 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 Merchandise__c, and a new Merchandise__c record is inserted that fires the two triggers, the order in which these triggers fire isn't guaranteed.
- `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 will contain 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

Triggers are only invoked for data manipulation language (DML) operations that are initiated or processed by the Java application server. Consequently, some system bulk operations don't currently 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

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 of 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.

# 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`

### 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

#### Additional Considerations for Chatter Objects

Things to consider about FeedItem and FeedComment triggers:

- Only FeedItems of `Type` `TextPost`, `LinkPost`, and `ContentPost` 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 information is saved, which means that `ConnectApi.FeedItem.attachment` information may not be available in the trigger.

  The attachment information may not be available from these methods: `ConnectApi.ChatterFeeds.getFeedItem`, `ConnectApi.ChatterFeeds.getFeedPoll`, `ConnectApi.ChatterFeeds.postFeedItem`, `ConnectApi.ChatterFeeds.shareFeedItem`, and `ConnectApi.ChatterFeeds.voteOnFeedPoll`.
- 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.

## 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 327.

If a trigger ever throws an unhandled exception, all records are marked with an error and no further processing takes place.

**See Also:**

*addError(String)*
*field.addError(String)*

## 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. This doesn't support bulk operations invoked through SOAP API.

```
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 Understanding Execution Governors and Limits on page 203.

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.new.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

## 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 defined in the `Util` class performs the insertion of a user with a role. The main method inserts an account and calls this future method.

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 10 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 is an organization-wide limit and is shared with all other asynchronous Apex: batch Apex and scheduled Apex.

**Note:** Future method jobs queued before a Database.com 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());
    }
}
```

## 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 Database.com user interface, or the `System.schedule` method.

**Important:** Database.com 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 Database.com 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 100 that are allowed. 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.

You cannot update an Apex class if there are one or more active scheduled jobs for that class.

### Implementing the `Schedulable` Interface

To schedule an Apex class to run at regular intervals, first write an Apex class that implements the Database.com-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 Database.com 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.com 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));

      Merchandise__c a = [SELECT Id, Name FROM Merchandise__c WHERE Name =
                  'Merchandise A'];
      a.name = 'Updated Merchandise';
      update a;
   }
}
```

The following tests the above class:

```
@istest
class TestClass {

   static testmethod void test() {
   Test.startTest();

      Merchandise__c a = new Merchandise__c();
      a.Name = 'Merchandise A';
      a.Description__c='Office supplies';
      a.Price__c=1.25;
      a.Total_Inventory__c=100;
      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('Updated Merchandise',
         [SELECT id, name FROM Merchandise__c WHERE id = :a.id].name);

   Test.stopTest();

   System.assertEquals('Updated Merchandise',
   [SELECT Id, Name FROM Merchandise__c 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 100 that are allowed. 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:** Database.com 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:<br>• JAN<br>• FEB<br>• MAR<br>• APR<br>• MAY<br>• JUN<br>• JUL<br>• AUG<br>• SEP<br>• OCT<br>• NOV<br>• DEC | , - * / |
| *Day_of_week* | 1–7 or the following:<br>• SUN<br>• MON<br>• TUE<br>• WED<br>• THU<br>• FRI<br>• 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, |

| Special Character | Description |
|---|---|
| | 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, 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.<br>  **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 Database.com 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 is an organization-wide limit and is shared with all other asynchronous Apex: batch Apex and future methods.

### Apex Scheduler Best Practices

- Database.com 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 100 that are allowed. 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.
- 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 Database.com 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 on Database.com. 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 the dataon 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 Database.com 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 five that are allowed. 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 Database.com user interface. For more information on the Schedule Apex page, see “Scheduling Apex” in the Database.com 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 179.

For more information on Apex managed sharing, see Understanding Apex Managed Sharing on page 142.

## Using Batch Apex

To use batch Apex, you must write an Apex class that implements the Database.com-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 be passed to the interface method `execute`. This method returns either a `Database.QueryLocator` object or an iterable that contains the records or objects being 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 Merchandise__c object can return a `QueryLocator` for all merchandise records (up to 50 million records) in an organization. Another example is a sharing recalculation for the Invoice_Statement__c object that returns a `QueryLocator` for all invoice statement records in an organization.

Use the iterable when you need 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`, the returned list should be used.

Batches of records execute in the order they are received from the `start` method.

- `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 of 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
      FROM AsyncApexJob WHERE Id =
      :BC.getJobId()];
   Integer i = a.TotalJobItems;
   Integer j = a.NumberOfErrors;
}
```

### Using Database.QueryLocator to Define Scope

The `start` method can return either a `Database.QueryLocator` object that contains the records to be used 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 be used 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 CustomInvoiceIterable();
   }
   global void execute(Database.BatchableContext info,
                       List<Invoice_Statement__c> scope){
       List<Invoice_Statement__c> invsToUpdate =
                         new List<Invoice_Statement__c>();
       for(Invoice_Statement__c i : scope){
           i.Name = 'true';
           i.NumberOfEmployees = 70;
           invsToUpdate.add(i);
       }
       update invsToUpdate;
   }
   global void finish(Database.BatchableContext info){
   }
}
```

### Using the `Database.executeBatch` Method

You can use the `Database.executeBatch` method to programmatically begin a batch job.

**Important:** When you call `Database.executeBatch`, Database.com only adds the process to the queue. Actual execution may be delayed based on service availability.

The `Database.executeBatch` method takes two parameters:

- An instance of a class that implements the `Database.Batchable` interface.
- The `Database.executeBatch` method takes an optional parameter *`scope`*. This parameter specifies the number of records that should be passed 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 thereby 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, Database.com 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 may run into other limits.

The `Database.executeBatch` method returns the ID of the AsyncApexJob object, which can then be used 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 ];
```

For more information about the AsyncApexJob object, see AsyncApexJob in the *Object Reference for Database.com.*

You can also use this ID with the `System.abortJob` method.

### 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 should start executing.
- An optional scope value. This parameter specifies the number of records that should be passed 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 thereby 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, Database.com 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 may 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 call 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 about CronTrigger, see CronTrigger in the *Object Reference for Database.com*.

**Note:** Some things to note about `System.scheduleBatch`:

- When you call `System.scheduleBatch`, Database.com 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.
- All scheduled Apex limits apply for batch jobs scheduled using `System.scheduleBatch`. After the batch job starts executing, 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 UpdateInvoiceFields implements Database.Batchable<sObject>{
   global final String Query;
   global final String Entity;
   global final String Field;
   global final String Value;

   global UpdateInvoiceFields(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){

    }

}
```

The following code can be used to call the above class:

```
// Query for 10 invoice statements
String q = 'SELECT Description__c FROM Invoice_Statement__c 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 invoices 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<Invoice_Statement__c> invs = new List<Invoice_Statement__c>();

   for(sObject s : scope){
        Invoice_Statement__c a = (Invoice_Statement__c)s;
        if(a.OwnerId==fromUserId){
            a.OwnerId=toUserId;
            invs.add(a);
        }
   }

update invs;

}
global void finish(Database.BatchableContext BC){

}
}
```

Use the following to execute the `OwnerReassignment` class in the previous example:

```
OwnerReassignment reassign = new OwnerReassignment();
reassign.query = 'SELECT Id, Name, Ownerid ' +
                 'FROM Invoice_Statement__c ' +
                 '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 invoice statement records. The specified query selects invoice statements 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);
// Query for selecting the invoices to delete
BDel.query = 'SELECT Id FROM Invoice_Statement__c ' +
    'WHERE 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, you must specify `Database.AllowsCallouts` in the class definition. For example:

```
global class SearchAndReplace implements Database.Batchable<sObject>,
   Database.AllowsCallouts{
}
```

Callouts include HTTP requests as well as 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 invoice statement records. You could define a method in `execute` to aggregate totals of the invoice 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 the `Invoice_Value__c` invoice statement field as the records are processed:

```
global class SummarizeInvoiceTotal implements
   Database.Batchable<sObject>, Database.Stateful{

   global final String Query;
   global integer Summary;

   global SummarizeInvoiceTotal(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('Invoice_Value__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 Invoice statement 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 changed the value of `initialState` in `execute`, the second chunk of processed records would not be able to access the new value: only the initial value would be accessible.

### Testing Batch Apex

When testing your batch Apex, you can test only one execution of the `execute` method. You can 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 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.com API version 25.0 and later, but not in earlier versions.

Starting with Apex saved using Salesforce.com 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. You must place the `catch` block after the `stopTest` method. Note however that with Apex saved using Salesforce.com 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 example below 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'];
// Create 200 test invoice statements -
// this simulates one execute.
// Important - the Apex test framework only allows you to
// test one execute.

   List <Invoice_Statement__c> invs =
                        new List<Invoice_Statement__c>();
      for(Integer i = 0; i<200; i++){
         Invoice_Statement__c a =
                new Invoice_Statement__c(
                   Description__c ='Invoice '+'i',
                Ownerid = u.ID);
         invs.add(a);
      }

   insert invs;

   Test.StartTest();
   OwnerReassignment reassign = new OwnerReassignment();
   reassign.query='SELECT Id, Name, Ownerid ' +
            'FROM Invoice_Statement__c ' +
            '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 Invoice_Statement__c WHERE OwnerId=\'' + u2.Id + '\''),
      200);

   }
}
```

### Batch Apex Governor Limits

Keep in mind the following governor limits for batch Apex:

- Up to five queued or active batch jobs are allowed for Apex.
- 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 is an organization-wide limit and is shared with all other asynchronous Apex: scheduled Apex and future methods.
- 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 different limit of 5 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, Database.com 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 may run into other limits.
- If no size is specified with the optional *`scope`* parameter of `Database.executeBatch`, Database.com 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 five that are allowed. 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`, Database.com only places the job in the queue. Actual execution may be delayed based on service availability.
- When testing your batch Apex, you can test only one execution of the `execute` method. You can 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 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.
- Starting with Apex saved using Salesforce.com API version 26.0, you can call `Database.executeBatch` or `System.scheduleBatch` from the `finish` method. This enables you to start or schedule a new batch job when the current batch job finishes. For previous versions, you can't call `Database.executeBatch` or `System.scheduleBatch` from any batch Apex method. Note that the version 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 Salesforce.com API version of the helper class doesn't matter.
- When a batch Apex job is run, email notifications are sent either to the user who submitted 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 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 Database.com.*
- For each 10,000 `AsyncApexJob` records, Apex creates one additional `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 will return one more record for every 10,000 `AsyncApexJob` records. For more information about the `AsyncApexJob` object, see AsyncApexJob in the *Object Reference for Database.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. This ensures the sharing is accurate and complete.
- Batch jobs queued before a Database.com 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.

**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 Database.com. Use the `webService` keyword to define these methods. For example:

```
global class MyWebService {
    webService static Id createInvoiceStatement(String description) {
        Invoice_Statement__c inv = new Invoice_Statement__c(Description__c = description);
        insert inv;
        return inv.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 because you cannot define a method 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`.
- 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

Considerations for calling Apex SOAP Web service methods:

- 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 WarehouseService {

  global class InvoiceInfo {
     webService String Description;
  }

  webService static Invoice_Statement__c createInvoice(InvoiceInfo info) {
```

```
    Invoice_Statement__c inv = new Invoice_Statement__c();
    inv.Description__c = info.Description;
    insert inv;
    return inv;
  }
}
```

```
// Test class for the previous class.
@isTest
private class WarehouseServiceTest {
  testMethod static void testInvoiceCreate() {
    WarehouseService.InvoiceInfo info = new WarehouseService.InvoiceInfo();
    info.Description = 'My Web invoice';
    Invoice_Statement__c inv = WarehouseService.createInvoice(info);
    System.assert(inv != null);
  }
}
```

You can invoke this Web service using AJAX. For more information, see Apex in AJAX on page 199.

### 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 76

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 3 MB. For more information, see Understanding 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`.
- 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>
```

- 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. |
| 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

This code sample shows 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 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='/Invoice_Statement__c/*')
global with sharing class MyRestResource {

    @HttpDelete
    global static void doDelete() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String invId = req.requestURI.substring(
                                req.requestURI.lastIndexOf('/')+1);
        Invoice_Statement__c inv =
                         [SELECT Id FROM Invoice_Statement__c
                          WHERE Id = :invId];
        delete inv;
    }

    @HttpGet
    global static Invoice_Statement__c doGet() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String invId = req.requestURI.substring(
                                 req.requestURI.lastIndexOf('/')+1);
        Invoice_Statement__c result =
                       [SELECT Id, Description__c
                        FROM Invoice_Statement__c
                        WHERE Id = :invId];
        return result;
    }
```

```
  @HttpPost
    global static String doPost(String status,
        String description) {
        Invoice_Statement__c inv = new Invoice_Statement__c();
        inv.Status__c = status;
        inv.Description__c = description;
        insert inv;
        return inv.Id;
    }
}
```

2. To call the `doGet` method from a client, open a command-line window and execute the following `cURL` command to retrieve an invoice statement by ID:

   ```
   curl -H "Authorization: Bearer sessionId"
   "https://instance.salesforce.com/services/apexrest/Invoice_Statement__c/invoiceId"
   ```

   - Replace ***sessionId*** with the `<sessionId>` element that you noted in the login response.
   - Replace ***instance*** with your `<serverUrl>` element.
   - Replace ***invoiceId*** with the ID of an invoice statement which exists in your organization.

   After calling the `doGet` method, Database.com returns a JSON response with data such as the following:

```
{
  "attributes" :
    {
      "type" : "Invoice_Statement__c",
      "url" : "/services/data/v22.0/sobjects/Invoice_Statement__c/invoiceId"
    },
  "Id" : "invoiceId",
  "Description__c" : "Invoice 1"

}
```

**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 `invoice.txt` to contain the data for the invoice statement you will create in the next step.

```
{
  "description" : "My invoice",
  "status" : "Open"
}
```

4. Using a command-line window, execute the following `cURL` command to create a new invoice statement:

   ```
   curl -H "Authorization: Bearer sessionId" -H "Content-Type: application/json" -d
   @invoice.txt "https://instance.salesforce.com/services/apexrest/Invoice_Statement__c/"
   ```

   After calling the `doPost` method, Database.com returns a response with data such as the following:

```
"invoiceId"
```

   The ***invoiceId*** is the ID of the invoice statement you just created with the POST request.

5. Using a command-line window, execute the following `cURL` command to delete an invoice statement by specifying the ID:

```
curl –X DELETE –H "Authorization: Bearer sessionId"
"https://instance.salesforce.com/services/apexrest/Invoice_Statement__c/invoiceId"
```

### 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.

# Invoking Apex Using JavaScript

## 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 CreateInvoiceLineItem(
              Integer units, Decimal price, Invoice_Statement__c inv) {
      Line_Item__c i = new Line_Item__c(
                       Units_Sold__c=units,
                       Unit_Price__c=price,
                       Invoice_Statement__r=inv);
        return i.id;
    }
}
```

By using the following JavaScript code:

```
var invoice = sforce.sObject("Invoice_Statement__c");
var id = sforce.apex.execute("myClass","CreateInvoiceLineItem",
                             {units:"5",
                              price:"1.25",
                              inv:invoice});
```

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", {});
```

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.

## 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;
    }
}
```

# Understanding Execution Governors and Limits

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. If some Apex code ever exceeds a limit, the associated governor issues a runtime 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
- Force.com Platform Apex Limits
- Static Apex Limits
- Size-Specific Apex Limits
- Miscellaneous Apex Limits

## 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] | 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 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 | |

| Description | Synchronous Limit | Asynchronous Limit |
|---|---|---|
| Total number of callouts (HTTP requests or Web services calls) in a transaction | 10 | |
| Maximum timeout for all callouts (HTTP requests or Web services calls) in a transaction | 120 seconds | |
| Total number of methods with the `future` annotation allowed per Apex invocation | 10 | |
| Total number of describes allowed[4] | 100 | |
| Total heap size[5] | 6 MB | 12 MB |
| Maximum CPU time on the Database.com servers[6] | 10,000 milliseconds | 60,000 milliseconds |
| Maximum execution time for each Apex transaction | 10 minutes | |
| Maximum number of unique namespaces referenced[7] | 10 | |

[1] In a SOQL query with parent-child relationship sub-queries, each parent-child relationship counts as an additional 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] Describes include the following methods and objects.

- ChildRelationship objects
- RecordTypeInfo objects
- PicklistEntry objects
- `fields` calls

[5] Email services heap size is 36 MB.

[6] CPU time is calculated for all executions on the Database.com application servers occurring in one Apex transaction—for the executing Apex code, and 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.

[7] 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`.
- Use the Limits methods to determine the code execution limits for your code while it is running. For example, you can use the `getDMLStatements` method to determine the number of DML statements that have already been called by your program, or the `getLimitDMLStatements` method to determine the total number of DML statements available to your code.

## 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, 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 simultaneous requests to URLs with the same host for a callout request[3] | 20 |
| Maximum number of Apex classes scheduled concurrently | 100 |
| Maximum number of Batch Apex jobs queued or active | 5 |
| Maximum number of Batch Apex job `start` method concurrent executions[4] | 1 |
| Total number of test classes that can be queued per a 24-hour period[5] | The greater of 500 or 10 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 is an organization-wide limit and is shared with all asynchronous Apex: Batch Apex, scheduled Apex, and future methods.

[2] If additional requests are made while the 10 long-running requests are still running, they're denied.

[3] The host is defined by the unique subdomain for the URL, for example, `www.mysite.com` and `extra.mysite.com` are two different hosts. This limit is calculated across all organizations that access the same host. If this limit is exceeded, a `CalloutException` will be thrown.

[4] 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.

[5] This limit applies to tests running asynchronously. This includes tests started through the Database.com 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 Database.com features are tracked separately. For example, you can have 50 Apex query cursors, 15 cursors for the Batch Apex `start` method, and 5 cursors for the Batch Apex `execute` and `finish` methods each.

## 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] | 3 MB |
| Maximum SOQL query run time before the transaction can be canceled by Database.com | 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 belong 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, non-selective SOQL queries may be terminated by the system. Developers will 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.

**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.

# Using Governor Limit Email Warnings

When an end-user invokes Apex code that surpasses more than 50% of any governor limit, you can specify a user in your organization to receive an email notification of the event with additional details. To enable email warnings:

1. Log in to Database.com as an administrator user.
2. From Setup, click **Manage Users** > **Users**.
3. Click **Edit** next to the name of the user who should 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 Database.com Features with Apex

**In this chapter ...**



Several Database.com 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.

# Working with Chatter in Apex

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*.

Use Chatter in Apex to develop native, social Database.com applications. Create feeds, post feed items with mentions and topics, and update user and group photos.

In Apex, it is 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 logged-in 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 339.

### Chatter in Apex Quick Start

This quick start shows you how to get started with Chatter in Apex. Follow the steps to use Chatter in Apex to display the Chatter feeds of two groups side by side in a Visualforce page.

### Working with Feeds and Feed Items

Feeds are made up of feed items. A feed item is a piece of information posted by a user (for example, a poll) or by an automated process (for example, when a tracked field is updated on a record). Because feeds and feed items are the core of Chatter, understanding them is crucial to developing applications with Chatter REST API and Chatter in Apex.

### 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.

### Accessing ConnectApi Data in Communities and Portals

Most `ConnectApi` methods work within the context of a single community.

### 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*

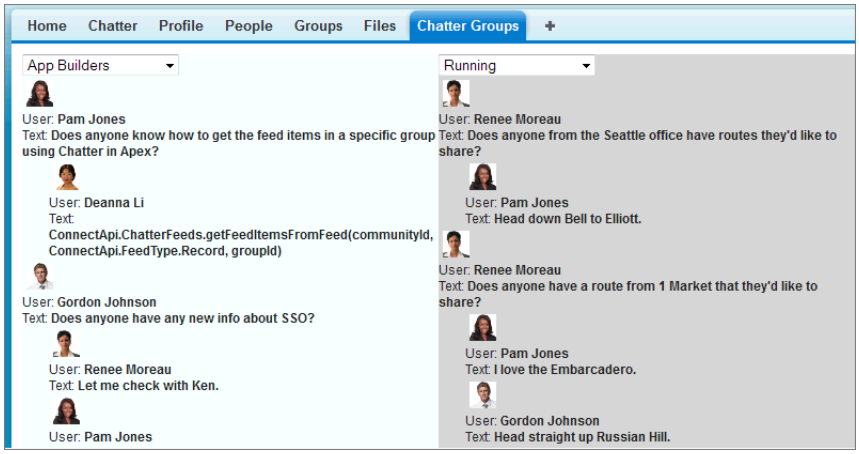
Please be aware of these additional differences between `ConnectApi` classes and other Apex classes.

## Chatter in Apex Quick Start

This quick start shows you how to get started with Chatter in Apex. Follow the steps to use Chatter in Apex to display the Chatter feeds of two groups side by side in a Visualforce page.

**Tip:** You can also watch a video of this quick start: Using Chatter in Apex to Display Two Chatter Feeds in a Visualforce Page. The video displays the news feeds of two Salesforce Communities instead of two groups feeds, but the code is very similar.

Create an Apex controller that uses Chatter in Apex to populate a drop-down list with the groups that the logged-in user is a member of. The controller code then takes the selected group and gets first page of feed items for that group. Next, create a custom Visualforce component to display the feed items. Finally, create a Visualforce page that contains two instances of the custom component:




This quick start is designed to get you up and running with Chatter in Apex as quickly as possible so the user interface is very simple. In a real-world scenario, you would customize the user interface to match your organization's branding.

### *Prerequisites*

To complete the quick start you need access to a Developer Edition organization. You must also create at least two groups and post to them so their feeds contain data.

### *Step 1: Get the Chatter Feed and Group Data*

The first step is to create an Apex controller that uses Chatter in Apex to populate a drop-down list with the Chatter groups that the logged-in user is a member of. The controller also uses Chatter in Apex to get the feed for the selected group.

### *Step 2: Display the Feed and Group Data in a Visualforce Component*

The second step is to create a Visualforce custom component called GroupFeed that displays the data from the Apex controller.

### *Step 3: Display the Component in a Visualforce Page*

The third step is to create a Visualforce page called DoubleGroupFeed that contains two instances of the GroupFeed custom component we created in step 2.

### *Step 4: Create a Chatter Groups Tab*

The final step is to create a tab in Salesforce.com that links to the DoubleGroupFeed Visualforce page.

## Prerequisites

To complete the quick start you need access to a Developer Edition organization. You must also create at least two groups and post to them so their feeds contain data.

You can't develop Apex in your Database.com 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 a test database organization.

## Step 1: Get the Chatter Feed and Group Data

The first step is to create an Apex controller that uses Chatter in Apex to populate a drop-down list with the Chatter groups that the logged-in user is a member of. The controller also uses Chatter in Apex to get the feed for the selected group.

1. Click *Your Name* > **Developer Console**.
2. In the Developer Console, click **File** > **New** > **Apex Class**.
3. Enter the name `GroupFeedController` and click OK.
4. Copy this code and paste it into the GroupFeedController class, replacing the existing code:

```
global class GroupFeedController{

    // Declare and assign values to strings to use as method parameters.
    private static String communityId = null;
    private static String userId = 'me';

    // Holds the ID of the selected group.
    // Pass this property to getFeedItemsFromFeed to get the group's feed.
    global String groupId { get; set; }

    // Get the IDs and names for all of the groups
    // the logged-in user is a member of. Add them to
    // a List of SelectionOption objects. This List populates
    // the drop-down menu in the GroupFeed custom component.
    global static List<SelectOption> getGroupOptions() {
        List<SelectOption> options = new List<SelectOption>();

        // Adds a blank option to display when the page loads.
        options.add(new SelectOption('', ''));

        // Declare and assign values to strings to use as method parameters.
        Integer page = 0;
        Integer pageSize = 100;

        // Use Chatter in Apex to get the names and IDs of every group
        // the logged-in user is a member of.
        // Chatter in Apex classes are in the ConnectApi namespace.
        // communityId -- a community ID or null.
        // userId -- the user ID or the keyword 'me' to specify the logged-in user.
        // page -- the page number to return.
        // pageSize -- the number of items on the page.
      ConnectApi.UserGroupPage groupPage = ConnectApi.ChatterUsers.getGroups(communityId,
userId, page, pageSize);

        // The total number of groups the logged-in user is a member of.
        Integer total = groupPage.total;

        // Loop through all the groups and add each group's id and name
        // to the list of selection options.
        while (page * pageSize < total) {
```

```
            // groupPage.groups is a List of ConnectApi.ChatterGroupSummary objects.
            // ChatterGroupSummary is a subclass of ChatterGroup.
            // For each ChatterGroup object in the List...
            for (ConnectApi.ChatterGroup grp : groupPage.groups) {
                // Add the group's ID and name to the list of selection options.
                options.add(new SelectOption(grp.id, grp.name));
            }

            page++;

            if (page * pageSize < total) {
                // Get the next page of groups.
                groupPage = ConnectApi.ChatterUsers.getGroups(communityId, userId, page,
 pageSize);
            }
        }

        // Return the list of selection options.
        return options;
    }

    // Get the feed items that make up a group's feed.
    global List<ConnectApi.FeedItem> getFeedItems() {
        if (String.isEmpty(groupId)) { return null; }
        // To get the feed for a group, use the Record feed type and pass a group ID.
        // getFeedItemsFromFeed returns a ConnectApi.FeedItemPage class.
        // To get the List of ConnectApi.FeedItem objects,
        // add the .items property to the call.
        return ConnectApi.ChatterFeeds.getFeedItemsFromFeed(communityId,
ConnectApi.FeedType.Record, groupId).items;
    }

    public PageReference choose() {
        return null;
    }

}
```

5. Click **File** > **Save**.

## Step 2: Display the Feed and Group Data in a Visualforce Component

The second step is to create a Visualforce custom component called GroupFeed that displays the data from the Apex controller.

1. In the Developer Console, click **File** > **New** > **Visualforce Component**.
2. Enter the name `GroupFeed` and click OK.
3. Copy this code and paste it into the GroupFeed component, replacing the existing code:

```
<apex:component controller="GroupFeedController">

    <!-- Display the drop-down list of group names. -->
    <apex:form >
        <!-- Bind the selection value to the groupId property in the controller. -->
        <apex:selectList value="{!groupId}" size="1">
            <!-- Get the selection options from the getGroupOptions method in the
controller. -->
            <apex:selectOptions value="{!groupOptions}"/>
            <apex:actionSupport event="onchange" rerender="feed"/>
        </apex:selectList>
    </apex:form>

    <!-- Display the feed for the selected group. -->
    <apex:outputPanel id="feed">
        <!-- Display the feed items.
             Call the getFeedItems method in the controller to get the List of FeedItem
 objects to display.
```

```
             Use the feedItem var to reference a FeedItem object in the List. -->
        <apex:repeat value="{!feedItems}" var="feedItem">
          <div>
            <!-- Display the photo for the feed item, the name of the actor who posted
the feed item,
                 and the text of the feed item. -->
            <apex:image style="margin:4px" width="25" url="{!feedItem.photoUrl}"/><br/>
            User: <b>{!feedItem.actor.name}</b><br/>
            Text: <b>{!feedItem.body.text}</b><br/>

            <apex:outputPanel >
              <!-- Display the comments on the feed item.
                   Use the reference to the FeedItem object on line 17
                   to get the List of ConnectApi.Comment objects to display.
                   Use the comment var to reference a Comment object in the List. -->
              <apex:repeat value="{!feedItem.comments.comments}" var="comment">
                 <div style="margin-left:25px">
                   <!-- Display the photo and name of the user who commented,
                        and display the text of the comment. -->
                   <apex:image style="margin:4px" width="25"
url="{!comment.user.photo.smallPhotoUrl}"/><br/>
                   User: <b>{!comment.user.name}</b><br/>
                   Text: <b>{!comment.body.text}</b>
                 </div>
              </apex:repeat>
            </apex:outputPanel>
          </div>
        </apex:repeat>
    </apex:outputPanel>
</apex:component>
```

4. Click **File** > **Save**.

## Step 3: Display the Component in a Visualforce Page

The third step is to create a Visualforce page called DoubleGroupFeed that contains two instances of the GroupFeed custom component we created in step 2.

1. In the Developer Console, click **File** > **New** > **Visualforce Page**.
2. Enter the name `DoubleGroupFeed` and click OK.
3. Copy this code and paste it into the DoubleGroupFeed page, replacing the existing code:

```
<apex:page sidebar="false" >
     <!-- Display two GroupFeed components side by side. -->
     <div id="column1-wrap" style="float:left; width:50%; background-color:AliceBlue">
          <div id="column1"><c:GroupFeed /></div>
     </div>
     <div id="column2" style="float:right; width:50%;
background-color:LightGray"><c:GroupFeed /></div>
</apex:page>
```

The `<div>` HTML elements create two vertical columns on the page.

4. Click **File** > **Save**.

## Step 4: Create a Chatter Groups Tab

The final step is to create a tab in Salesforce.com that links to the DoubleGroupFeed Visualforce page.

1. In Salesforce.com, from Setup, click **Create** > **Tabs**.
2. Click **New** in the Visualforce Tabs related list.
3. Select the DoubleGroupFeed Visualforce page to display in the custom tab.

4. Enter the label `Chatter Groups` to display on the tab.
5. Click the `Tab Style` lookup icon to display the Tab Style Selector. Click a tab style to select the color scheme and icon for the custom tab and click **Next**.
6. Select the tab visibility for each profile, or accept the default and click **Next**.
7. Specify the custom apps that should include the new tab, or accept the default and click **Save**.
8. Click the Chatter Groups tab to open the new page. Select groups from the drop-down lists to see their feeds.

   If you didn't create groups and post to them before you started, you won't see any content on the Chatter Groups page.

## Working with Feeds and Feed Items

Feeds are made up of feed items. A feed item is a piece of information posted by a user (for example, a poll) or by an automated process (for example, when a tracked field is updated on a record). Because feeds and feed items are the core of Chatter, understanding them is crucial to developing applications with Chatter REST API and Chatter in Apex.

**Note:** Salesforce Help refers to feed items as *posts*.

### How the Salesforce UI Displays Feed Items

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, regardless of the feed item type. The layout always contains the same elements and the elements are always in the same position; only the content of the layout elements changes. If you stick to this structure, you won't have to create a unique layout for every feed item type.

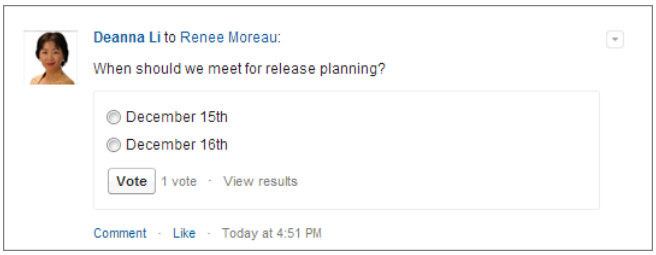

These are the feed item layout elements:

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. Preamble (`ConnectApi.FeedItem.preamble`)—Provides context. The same feed item can have a different preamble 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 preamble of the feed item in the group feed would be "Gordon Johnson (originally posted by Gordon Johnson)" and the "originally posted" text would link to the feed item on Gordon's profile.
3. Body (`ConnectApi.FeedItem.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 `text` property of the feed item's preamble, which always contains a value.
4. Auxiliary Body (`ConnectApi.FeedItem.attachment`)—The visualization of the attachment. There are multiple attachment types. For example, for a link post, the attachment is the link and name, for a poll, it's the poll data. In Chatter in Apex, the attachment types are subclasses of `ConnectApi.FeedItemAttachment`. In Chatter REST API, the

attachment types are exposed as response bodies with the name Feed Item Attachment: *Name*, for example, Feed Item Attachment: Link and Feed Item Attachment: Poll. Make sure your code has a default case that doesn't display an auxiliary body if it doesn't recognize the attachment type.

5. Created By Timestamp (`ConnectApi.FeedItem.relativeCreatedDate`)—The date and time when the feed item was posted. If the feed item is less then 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:




### Feed Item Visibility

The feed items 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 item, the user can see the feed item. Typically, a user sees feed updates for:

- Feed items that @mention the user if the user can access the feed item's parent
- Feed items 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 items posted to the user
- Feed items posted to groups the user owns or is a member of
- Feed items for standard and custom records, for example, tasks, events, leads, accounts, files, and so on

### Feed Types

There are many types of feeds. Each feed type is an algorithm that defines a collection of feed items.

**Important:** The algorithms, and therefore the collection of feed items, can change between releases.

In Chatter REST API, the feed types are exposed in the resources. For example, these are the resources for the news feed and topics feed:

```
/chatter/feeds/news/userId
/chatter/feeds/topics/topicId
```

In Chatter in Apex, all feed types except Filter and Favorites are exposed in the `ConnectApi.FeedType` enum and passed to the `ConnectApi.ChatterFeeds.getFeedItemsFromFeed` method or to the `ConnectApi.ChatterFeeds.postFeedItem` method. This example gets the feed items from the logged-in user's news feed and topics feed:

```
ConnectApi.FeedItemPage newsFeedItemPage =
   ConnectApi.ChatterFeeds.getFeedItemsFromFeed(null,
     ConnectApi.FeedType.News, 'me');

ConnectApi.FeedItemPage topicsFeedItemPage =
   ConnectApi.ChatterFeeds.getFeedItemsFromFeed(null,
     ConnectApi.FeedType.Topics, '0TOD00000000dUg');
```

To get a filter feed, call `ConnectApi.ChatterFeeds.getFeedItemsFromFilterFeed`. To get a favorites feed, call `ConnectApi.ChatterFavorites.getFeedItems`.

These are the feed types and their descriptions:

- `Bookmarks`—Contains all feed items saved as bookmarks by the logged-in 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 logged-in 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 logged-in user either owns or is a member of.
- `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 logged-in user follows, groups the user is a member of, files and records the user is following, all updates for records whose parent is the logged-in user, and every feed item and comment that mentions the logged-in user or that mentions a group the logged-in user is a member of.
- `People`—Contains all feed items posted by all people the logged-in 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.
- `To`—Contains all feed items with mentions of the logged-in user, feed items the logged-in user commented on, and feed items created by the logged-in 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 logged-in user. Favorites are feed searches, list views, and topics.
- `Filter`—Contains the news feed filtered to contain items whose parent is a specified entity type.

### Posting a Feed Item

Use these resources and methods to post feed items:

| Feed Type | Chatter REST API Resource | Chatter in Apex Method |
|---|---|---|
| News Feed | `POST /chatter/feeds/news /`***`userId`***`/feed-items`<br>***`userId`*** must be the ID of the logged-in user or the alias `me`. | `ConnectApi.ChatterFeeds.postFeedItem (`***`communityIdOrNull`***`, ConnectApi.FeedType.News, `***`userId`***`)`<br>***`userId`*** must be the ID of the logged-in user or the alias `me`. |
| Record Feed | `POST /chatter/feeds/record /`***`recordId`***`/feed-items` | `ConnectApi.ChatterFeeds.postFeedItem (`***`communityIdOrNull`***`, ConnectApi.FeedType.Record, `***`recordId`***`)` |
| User Profile Feed | `POST /chatter/feeds/user-profile /`***`userId`***`/feed-items` | `ConnectApi.ChatterFeeds.postFeedItem (`***`communityIdOrNull`***`, ConnectApi.FeedType.UserProfile, `***`userId`***`)` |

When you post a feed item, you're creating a child of a standard or custom object. For Chatter REST API, specify the parent object in the ***`userId`*** or ***`recordId`*** section of the resource. For Chatter in Apex, specify the parent object in the in the ***`userId`*** or ***`recordId`*** argument.

The `parent` property of the posted feed item contains information about the parent object.

Select the correct feed type and parent object for the task you want to complete:

**Post to yourself**

Make a POST request to the news feed, the record feed, or the user profile feed.

For ***userId***, specify the user ID of the logged-in user or the alias `me`.

The `parent` property of the newly posted feed item contains the User Summary object (`ConnectApi.UserSummary`) of the logged-in user.

**Post to another user**

Make a POST request to the record feed or the user profile feed.

For ***recordId*** or ***userId***, specify the user ID of the target user.

The `parent` property of the newly posted feed item contains the User Summary object (`ConnectApi.UserSummary`) of the target user.

**Post to a group**

Make a POST request to the record feed.

For ***recordId***, specify the group ID.

The `parent` property of the newly posted feed item contains the Group object (`ConnectApi.ChatterGroupSummary`) of the specified group.

**Post to a record (such as a file or an account)**

Make a POST request to the record feed.

For ***recordId***, specify the record ID.

The `parent` property of the new feed item depends on the record type specified in ***recordId***. If the record type is File, the parent is the File Summary object (`ConnectApi.FileSummary`). If the record type is Group, the parent is a Group object (`ConnectApi.ChatterGroupSummary`). If the record type is User, the parent is a User Summary object (`ConnectApi.UserSummary`). For all other record types, the parent is a Record Summary object (`ConnectApi.RecordSummary`).

### Getting Feed Items from a Feed

Getting feed items from a feed is similar, but not identical, for each feed type.

To get the feed items from the company feed or the moderation feed, you don't need to specify a subject ID:

| Feed Type | Chatter REST API Resource | Chatter in Apex Method |
|---|---|---|
| Company | `GET /chatter/feeds/company/feed-items` | `ConnectApi.ChatterFeeds .getFeedItemsFromFeed (communityIdOrNull, ConnectApi.FeedType.Company)` |
| Moderation | `GET /connect/communities/communityId /chatter/feeds/moderation/feed-items` | `ConnectApi.ChatterFeeds .getFeedItemsFromFeed (communityIdOrNull, ConnectApi.FeedType.Moderation)` |

To get the feed items from the favorites and filter feeds you need to specify a *favoriteId* or a *keyPrefix*. The *keyPrefix* indicates the object type and is the first three characters of the object ID. For these feeds, the *subjectId* must be the ID of the logged-in user or the alias `me`.

| Feed Type | Chatter REST API Resource | Chatter in Apex Method |
|---|---|---|
| Favorites | `GET /chatter/feeds/favorites/`***`subjectId`***`/`***`favoriteId`***`/feed-items` | `ConnectApi.ChatterFavorites.getFeedItems(`***`communityIdOrNull, subjectId, favoriteId`***`)` |
| Filter | `GET /chatter/feeds/filter/`***`subjectId`***`/`***`keyPrefix`***`/feed-items` | `ConnectApi.ChatterFeeds.getFeedItemsFromFilterFeed(`***`communityIdOrNull, subjectId, keyPrefix`***`)` |

To get the feed items from a record feed you need to specify a record ID.

| Feed Type | Chatter REST API Resource | Chatter in Apex Method |
|---|---|---|
| Record | `GET /chatter/feeds/record/`***`recordId`***`/feed-items` | `ConnectApi.ChatterFeeds.getFeedItemsFromFeed(`***`communityIdOrNull`***`, ConnectApi.FeedType.Record, `***`recordId`***`)` |

**Tip:** The ***`recordId`*** 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.

To get the feed items from all other feed types you need to specify a subject ID. Replace the *`feedType`* to specify a different feed. For all the feed types in this table except the user profile feed and the topics feed, the *`subjectId`* must be the ID of the logged-in user or the alias `me`.

| Feed Type | Chatter REST API Resource | Chatter in Apex Method |
|---|---|---|
| Bookmarks, Files, Groups, News, People, To, Topics, User Profile | `GET /chatter/feeds/`***`feedType`***`/`***`subjectId`***`/feed-items`<br>For example: `GET /chatter/feeds/news/me/feed-items` | `ConnectApi.ChatterFeeds.getFeedItemsFromFeed(`***`communityIdOrNull, feedType, subjectId`***`)`<br>For example: `ConnectApi.ChatterFeeds.getFeedItemsFromFeed(`***`communityIdOrNull`***`, ConnectApi.FeedType.News, 'me')` |

**See Also:**

*ChatterFavorites Class*
*ChatterFeeds Class*

## 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, `ActorWithId` contains simple properties such as `id` and `url`, and also a sub-object, `reference`.

**See Also:**

*ConnectApi Input Classes*
*ConnectApi Output 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. Use `null` for this parameter.

Most URLs returned in `ConnectApi` output objects are Chatter REST API resources.

If you specify `null`, URLs returned in the output use one of these formats:

```
/chatter/resource

/connect/resource
```

## 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 and feed item attachments. Message segments in a feed item are typed as `ConnectApi.MessageSegment`. Feed item attachments are typed as `ConnectApi.FeedItemAttachment`. 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 ConnectApi.AbstractRecordField Class, ConnectApi.FeedItemAttachment Class, and ConnectApi.MessageSegment 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-items?q=chat*
```

This example is a Chatter in Apex method call:

```
ConnectApi.ChatterFeeds.searchFeedItems(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 issues:

- 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* or the *logged-in* 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 method calls 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. A test method name is the regular method name with a `setTest` prefix. The test method has a signature (combination of arguments) to match every signature of the regular method. If the regular method has three overloads, the test method has three overloads. If a method requires a `setTest` method, the requirement is stated in the method's "Usage" section.

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 value that was registered with matching arguments 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 arguments, 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(String, String, ConnectApi.ChatterGroupPage)*
*Testing ConnectApi Code*
*setTestSearchGroups(String, String, Integer, Integer, ConnectApi.ChatterGroupPage)*
*Testing ConnectApi Code*
*setTestSearchGroups(String, String, ConnectApi.GroupArchiveStatus, Integer, Integer, ConnectApi.ChatterGroupPage)*
*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* or the *logged-in* 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`.

# Publisher Actions

**Note:** In the application, QuickActions are referred to as actions or publisher actions.

The publisher actions feature lets you create actions and add them to the Chatter publisher on the home page, the Chatter tab, and record detail pages. It also allows you to customize the order in which the standard Chatter actions appear, including Post, File, Link, and Poll.

There are four general types of actions: create actions, log-a-call actions, update actions, and custom actions.

- *Create actions* let users create records. They're different from the Quick Create and Create New features on the 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 you define. For example, you might create a custom action to let users write comments longer than 5000 characters, or one that integrates a video conferencing application so 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 publisher 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*

# 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. Database.com prevents calls to unauthorized network addresses.

To learn more about the two types of callouts, see:

- SOAP Services: Defining a Class from a WSDL Document on page 228
- Invoking HTTP Callouts on page 239

**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. Database.com prevents calls to unauthorized network addresses.

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.

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, Database.com 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 will fail if the WSDL contains schema types or schema constructs that are not supported by Apex classes, or if the resulting classes exceed 1 million character limit on Apex classes. For example, the Database.com 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 class includes 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.

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 Understanding the Generated Code and add it to your organization.

***Understanding the Generated Code***
***Testing Web Service Callouts***
***Performing DML Operations and Mock Callouts***
***Considerations Using WSDLs***

## 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 Understanding the Generated Code on page 233:

### 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 Database.com 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 Database.com 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>
```

## Understanding the Generated Code

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://www.w3.org/2001/XMLSchema',
                            'string','0','1','false'};

        private String[] apex_schema_type_info = new String[]{
                            'http://doc.sample.com/docSample',
                            'true'};

        private String[] field_order_type_info = new String[]{
                            'EchoStringResult'};
    }

    public class DocSamplePort {

        public String endpoint_x = 'http://YourServer/YourService';

        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();
            docSample.EchoStringResponse_element response_x;
            request_x.input = input;
            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;
        }
    }

    public class EchoString_element {

        public String input;
        private String[] input_type_info = new String[]{
                                'input',
                                'http://www.w3.org/2001/XMLSchema',
                                'string','0','1','false'};
        private String[] apex_schema_type_info = new String[]{
                                'http://doc.sample.com/docSample',
                                'true'};
        private String[] field_order_type_info = new String[]{'input'};
    }
}
```

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.

The class generated above can be used to invoke external Web services. The following code shows how to call 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.

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 Understanding the Generated 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 first a test merchandise record, calls `Test.startTest`, sets the mock callout mode using `Test.setMock`, and 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
        Merchandise__c testMer = new Merchandise__c(
            Name='Pens',
            Description__c='Durable pens',
            Price__c=1.5,
            Total_Inventory__c=1000);
        insert testMer;

        // 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 249.
- Circular references in Apex classes are not allowed.
- More than one loopback connection to Database.com 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

Database.com 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 1073.

# 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.

## 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 getContent(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 Database.com user interface. To do this, log in to Database.com 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.

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 implementing the `HttpCalloutMock` interface.

To enable running DML operations before mock callouts in your test methods, see Performing DML Operations and Mock Callouts.

***Testing HTTP Callouts by Implementing the HttpCalloutMock Interface***
***Testing HTTP Callouts Using Static Resources***
***Performing DML Operations and Mock Callouts***

### 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.

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 Database.com 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`.

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 merchandise record, 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 CalloutClassTestDbcom {
     @isTest static void testCallout() {
        // Perform some DML to insert test data
        Merchandise__c testMer = new Merchandise__c(
            Name='Pens',
            Description__c='Durable pens',
            Price__c=1.5,
            Total_Inventory__c=1000);
        insert testMer;

        // 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 Database.com 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 Database.com 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 `wiki.developerforce.com/index.php/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.



## Generating Certificates

You can use a self-signed certificate generated in Database.com 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 Database.com 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 Database.com 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. |

| Field | Description |
|---|---|
| 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 Database.com 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 Database.com online help.

## Using Certificates with SOAP Services

After you have generated a certificate in Database.com, 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 Understanding the Generated 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 Database.com certificate because it does not follow security best practices for protecting private keys. When you use a Database.com certificate, the private key is not shared outside Database.com.

**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 Understanding the Generated Code on page 233.

```
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 Database.com, 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 10 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.

#### 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 Understanding the Generated Code on page 233. 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
```

# 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.
- 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 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.

### *Roundtrip Serialization and Deserialization*

Using the `JSON` class methods, you can perform roundtrip serialization and deserialization of your JSON content.

### *JSON Generator*

Using the `JSONGenerator` class methods, you can generate standard JSON-encoded content.

### *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.com API version of the Apex code saved.

**Serialization of queried sObject with additional fields set**

For Apex saved using Salesforce.com 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.com API version 28.0, the additional fields are included in the serialized JSON string.

**Serialization of aggregate query result fields**

For Apex saved using Salesforce.com 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.

**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 Database.com user interface. To do this, log in to Database.com 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 = 'http://www.cheenath.com/tutorial/sfdc/sample1/response.php';
        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.

***Reading and Writing XML Using Streams***

Apex provides classes for reading and writing XML content using streams.

***Reading and Writing XML Using the DOM***

Apex provides classes that enable you to work with XML content using the DOM (Document Object Model).

## 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.

***Reading XML Using Streams***

The XMLStreamReader class methods enable forward, read-only access to XML data.

***Writing XML Using Streams***

The XmlStreamWriter class methods enable the writing of XML data.

### 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="Database.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.

```
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 Database.com:

```
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 Database.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:

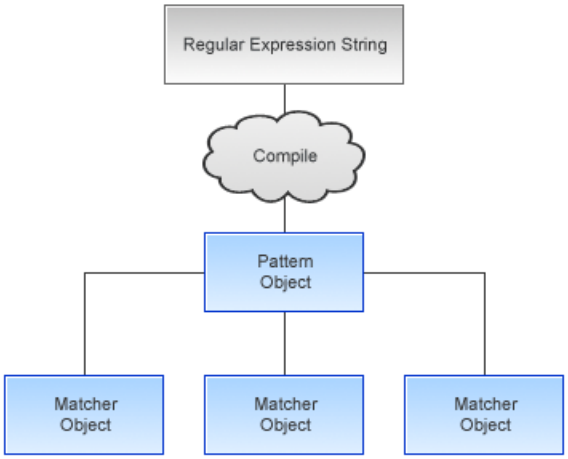



**Figure 4: Many Matcher objects can be created from the same Pattern object**

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:** Database.com 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');
```



## 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);
```

## See Also:

*Pattern Class*
*Matcher Class*

# FINISHING TOUCHES

## Chapter 12

## Debugging Apex

**In this chapter ...**

- Understanding the Debug Log
- Exceptions in Apex

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
  - 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;
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 |
| 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:

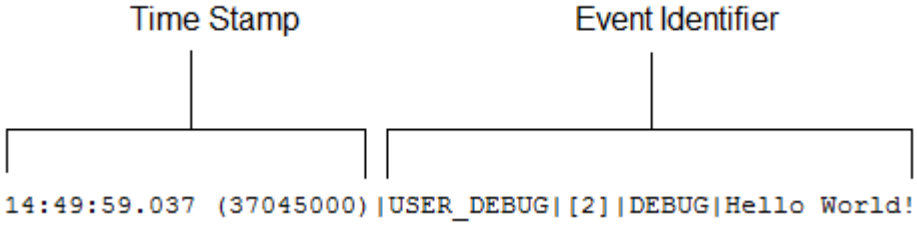
```
EXECUTION_STARTED
CODE_UNIT_STARTED|[EXTERNAL]execute_anonymous_apex
CODE_UNIT_STARTED|[EXTERNAL]MyTrigger on Merchandise trigger event BeforeInsert for
[new]
CODE_UNIT_FINISHED <-- The trigger ends
CODE_UNIT_FINISHED <-- The executeAnonymous ends
EXECUTION_FINISHED
```

Units of code include, but are not limited to, the following:

- Triggers
- Workflow invocations and time-based workflow
- Validation rules
- `@future` method invocations
- Web service invocations
- `executeAnonymous` calls
- 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

### 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.
- 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:

```
23.0
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|myTrigger on Merchandise
 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|myTrigger on Merchandise 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`.

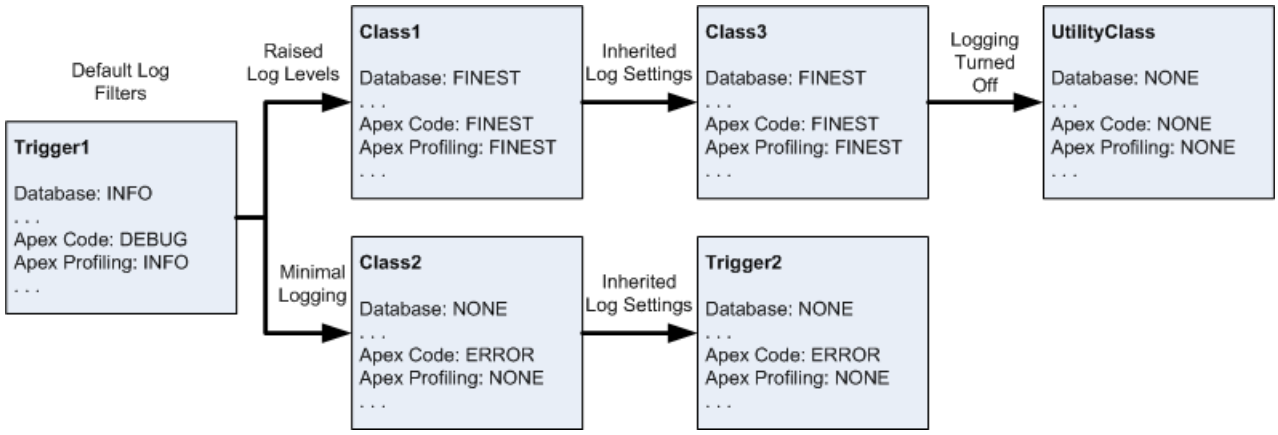



**Figure 5: Fine-tuning debug logging for classes and triggers**

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 Merchandise__c (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**.

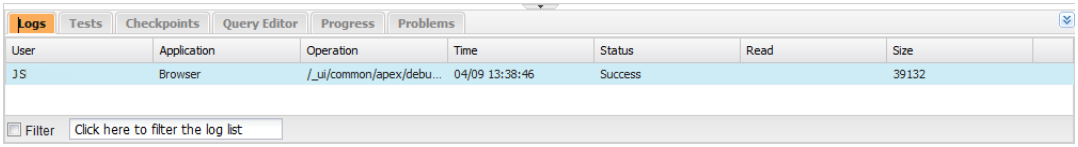
   The log filters are set to the default log levels.
3. Choose the log level desired for each log category.

To learn more about debug log categories, debug log levels, and debug log events, see Setting Debug Log Filters.

*Working with Logs in the Developer Console*
*Debugging Apex API Calls*

## 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 Database.com 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, 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 |

| Log Category | Description |
|---|---|
| | 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. |
| `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. If the Apex Code log level is set to FINEST, the deployment might 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.

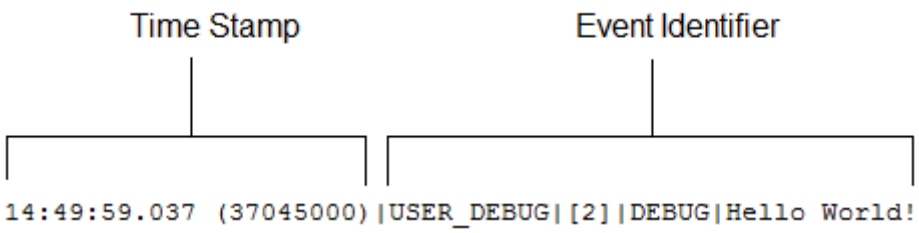



**Figure 6: Debug Log Line Example**

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.

```
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
```

**Figure 7: Debug Log Line Code Snippet**

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 |
| `CODE_UNIT_STARTED` | Line number and code unit name, such as `MyTrigger on Invoice_Statement__c 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 |
| `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 |
| `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 |

| Event Name | Fields or Information Logged with Event | Category Logged | Level Logged |
|---|---|---|---|
| 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_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_ELEMENT_FAULT | Message, Element type, and Element name (Element deferred) | 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 |
| 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 |
| LIMIT_USAGE_FOR_NS | Namespace and the following limits:<br>`Number of SOQL queries`<br>`Number of query rows`<br>`Number of SOSL queries`<br>`Number of DML statements`<br>`Number of DML rows` | Apex Profiling | FINEST |

| Event Name | Fields or Information Logged with Event | Category Logged | Level Logged |
|---|---|---|---|
| | `Number of code statements`<br>`Maximum heap size`<br>`Number of callouts`<br>`Number of Email Invocations`<br>`Number of fields describes`<br>`Number of record type describes`<br>`Number of child relationships describes`<br>`Number of picklist describes`<br>`Number of future calls`<br>`Number of find similar calls`<br>`Number of System.runAs() invocations` | | |
| `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.<br>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 |
| `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 |
| `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 |

| Event Name | Fields or Information Logged with Event | Category Logged | Level Logged |
|---|---|---|---|
| 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:<br>`var1:50`<br>`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:<br>`var1:50`<br>`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 |
| SYSTEM_CONSTRUCTOR_EXIT | Line number and the string `<init>()` with the types of parameters, if any, between the parentheses | System | DEBUG |
| SYSTEM_METHOD_ENTRY | Line number and method signature | System | DEBUG |
| SYSTEM_METHOD_EXIT | Line number and method signature | System | DEBUG |
| 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 |

| Event Name | Fields or Information Logged with Event | Category Logged | Level Logged |
|---|---|---|---|
| | | | 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 |
| VF_APEX_CALL | Element name, method name, and return type | Apex Code | INFO and above |
| VF_PAGE_MESSAGE | Message text | Apex Code | 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 |
| 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 |

| Event Name | Fields or Information Logged with Event | Category Logged | Level Logged |
|---|---|---|---|
| WF_FIELD_UPDATE | EntityName: NameField Id and the object or field name | Workflow | INFO 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 |
| 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:<br>• Db<br>• Workflow<br>• Validation<br>• Callout<br>• Apex_code |

| Element Name | Type | Description |
|---|---|---|
| | | • `Apex_profiling`<br>• `All` |
| `LogCategoryLevel` | `string` | Specifies the amount of information returned in the debug log. Only the `Apex_code` `LogCategory` uses the log category levels.<br>Valid log levels are (listed from lowest to highest):<br>• `ERROR`<br>• `WARN`<br>• `INFO`<br>• `DEBUG`<br>• `FINE`<br>• `FINER`<br>• `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:<br>• Variable declaration statements<br>• Start of loop executions<br>• All loop controls, such as break and continue<br>• Thrown exceptions *<br>• Static and class initialization code *<br>• Any changes in the with sharing context |

The corresponding output header, `DebuggingInfo`, contains the resulting debug log. For more information, see DebuggingHeader on page 1089.

# 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, Database.com sends an email to the developer with the exception information and the end user sees an error message in the Database.com user interface.

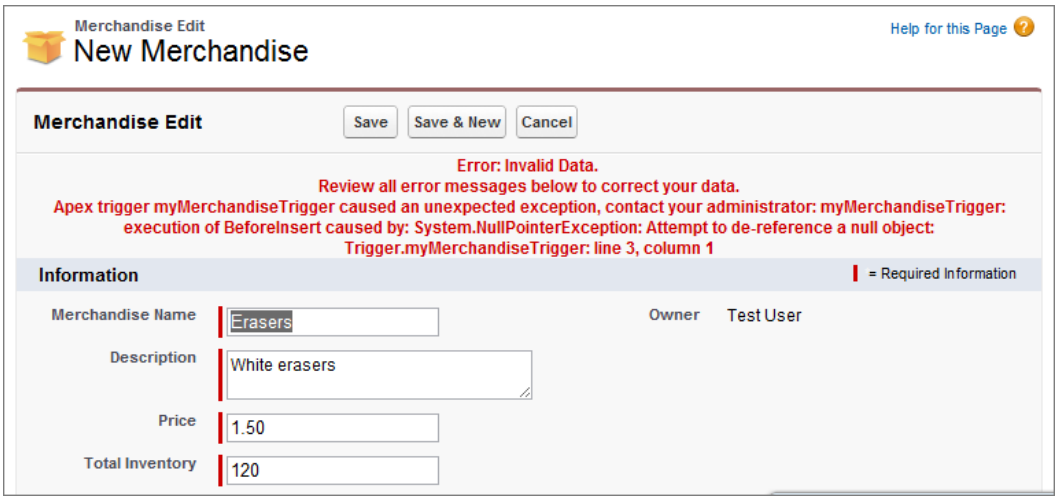
## Unhandled Exceptions Emails

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:** For Apex code that runs synchronously, some error emails may get suppressed for duplicate exception errors. For Apex code that runs asynchronously—batch Apex, scheduled Apex, or future methods (methods annotated with `@future`)—error 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 multiple associated `catch` statements; however, 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 optionally identifies a block of code that is guaranteed to execute and allows you to clean up after the code enclosed in the `try` block. A single `try` statement can have only one associated `finally` statement. Code in the `finally` block always executes regardless of the type of exception that was thrown and handled.

### Syntax

The syntax of these statements is as follows:

```
try {
 code_block
} catch (exceptionType) {
 code_block
}
// Optional catch statements for other exception types.
// Note that the general exception type, 'Exception',
// must be the last catch block when it is used.
} catch (Exception e) {
 code_block
}
// Optional finally statement
} 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 Database.com. 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 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.

### 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 Database.com 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.

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, 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 to a production organization, every unit test in your organization namespace is executed.
  ◊ 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.

Database.com 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.com 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 311.

## 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.com 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.
- 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.

**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.com 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.com 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.com API version 24.0 and later, test methods don't have access by default to pre-existing data in the organization, such as 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
- ApexClass
- ApexTrigger

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.com 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.com 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.com 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.
- For Apex saved using Salesforce.com 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. For example, 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.

## 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 merchandise item.
    // It also creates and accesses a new test merchandise item.
    static testmethod void myTestMethod1() {
        // Query an existing merchandise item in the organization.
        Merchandise__c m = [SELECT Id, Price__c, Total_Inventory__c, Description__c
                            FROM Merchandise__c WHERE Name='Pencils' LIMIT 1];
        System.assert(m != null);

        // Create a test merchandise item based on the queried merchandise item.
        Merchandise__c testMerchandise = m.clone();
        testMerchandise.Name = 'Test Pencil';
        insert testMerchandise;

        // Query the test merchandise that was inserted.
        Merchandise__c testMerchandise2 = [SELECT Id, Price__c, Total_Inventory__c
                            FROM Merchandise__c WHERE Name='Test Pencil' LIMIT 1];
        System.assert(testMerchandise2 != 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 invoice that is created here.
        Invoice_Statement__c inv = new Invoice_Statement__c(
                                   Description__c='Invoice 1');
        insert inv;
        // Access the invoice that was just created.
        Invoice_Statement__c insertedInv = [SELECT Id,Description__C
                                FROM Invoice_Statement__c
                                WHERE Description__c='Invoice 1'];
        System.assert(insertedInv != 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.

## 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;

        for (Integer j=0;j<numAccts;j++) {
            Account acct = accts[j];
            List<Contact> cons = new List<Contact>();
            // 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 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
    }
}
```

# Running Unit Test Methods

You can run unit tests for:

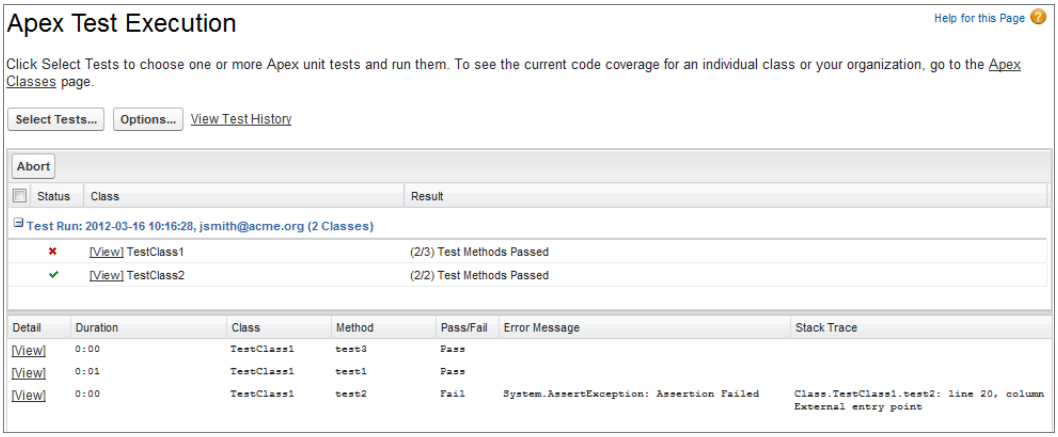
- A specific class
- A subset of classes
- All unit tests in your organization

To run a test, use any of the following:

- The Database.com user interface
- The Force.com IDE
- The Force.com Developer Console
- The API

## Running Tests Through the Database.com 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...**.
3. Select the tests to run. The list of tests includes only classes that contain test methods.

**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://wiki.developerforce.com/index.php/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 Database.com application from ***Your Name*** > **Developer Console**. For more details, check out the Developer Console documentation in the Database.com 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_database.com_server/services/wsdl/apex`, where *your_database.com_server* is equivalent to the server on which your organization is located, such as `<string_unique_to_your_org>.database.com`.

Though administrators in a Database.com production organization cannot make changes to Apex code using the Database.com 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. Database.com production organizations must use the `compileAndTest` SOAP API call to make changes to Apex code. For more information, see Deploying Apex on page 316.

For more information on `runTests()`, see SOAP API and SOAP Headers for Apex on page 1073.

## 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.

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);
            }
        }
    }
}
```

## 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.

## 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 Database.com 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
                                            Merchandise__c(Id, Name WHERE Name = 'test'
LIMIT 1)];
    }
}
```

Although the merchandise 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, 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 to a production organization, every unit test in your organization namespace is executed.
- 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 a `Merchandise__c` item A, and receive the ID `a0290000000UuSn`, then create another merchandise item B, the ID of item B may or may not be sequentially higher.
- On the list of Apex classes, there is a Code Coverage column. If you click the coverage percent number, a page displays, highlighting the lines of code for that class or trigger that are covered by tests in blue, as well as highlighting the lines of code that are not covered by tests in red. It also lists how many times a particular line in the class or trigger was executed by the test.
- 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 `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://wiki.developerforce.com/index.php/Apex_Toolkit_for_Eclipse` for details.
- In the Database.com 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.

# 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);
```

# Chapter 14

# Deploying Apex

**In this chapter ...**



You can't develop Apex in your Database.com 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 a test database 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 test database organization to your production organization. You can create an outbound change set in the Database.com 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 Database.com online help.

## 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.com to support its users and partners but isn't considered part of our services for purposes of the salesforce.com Master Subscription Agreement.

To deploy Apex from a local project in the Force.com IDE to a Database.com 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 to a production organization, every unit test in your organization namespace is executed.
  ◊ 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 test database organization to a production organization using Apache's Ant build tool.

**Note:** The Force.com Migration Tool is a free resource provided by salesforce.com to support its users and partners but isn't considered part of our services for purposes of the salesforce.com 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 Database.com 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. Copy the ant-salesforce.jar file from the unzipped file into the ant lib directory. The ant lib directory is located in the root folder of your Ant installation.
8. Open the sample subdirectory in the unzipped file.
9. Edit the build.properties file:
   a. Enter your Database.com 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 Database.com online help.

   b. If you are deploying to a test database 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 on `deploy`, see Understanding `deploy` on page 319.

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 `deploy` call completes successfully only if all of 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 to a production organization, every unit test in your organization namespace is executed.
  ◊ 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.

You cannot run more than one `deploy` Metadata API call at the same time.

The Force.com Migration Tool provides the task `deploy` which can be incorporated into your deployment scripts. You can modify the `build.xml` sample to include your organization's classes and triggers. The properties of the `deploy` task are as follows:

**`username`**

The username for logging into the Database.com production organization.

**`password`**

The password associated for logging into the Database.com production organization.

**`serverURL`**

The URL for the Database.com 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 test database. See Understanding retrieveCode on page 320 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`.

**runTests**

The name of the class that contains the unit tests that you want to run.

**Note:** This parameter is ignored when deploying to a Database.com production organization. Every unit test in your organization namespace is executed.

**runAllTests**

This property takes a Boolean value: `true` if you want run all tests in your organization, `false` if you do not. You should not specify a value for `runTests` if you specify `true` for `runAllTests`.

**Note:** This parameter is ignored when deploying to a Database.com production organization. Every unit test in your organization namespace is executed.

## Understanding `retrieveCode`

Use the `retrieveCode` call to retrieve classes and triggers from your test database 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 of the retrieve task are as follows:

**username**

The username for logging into the Database.com production organization.

**password**

The password associated for logging into the Database.com production organization.

**serverURL**

The URL for the Database.com server you are logging into. If you do not specify a value, the default is `www.salesforce.com`.

**apiversion**

Which version of the Metadata API at which the files should be retrieved.

**retrieveTarget**

The directory into which the files should be copied.

**unpackaged**

The name of file that contains the list of files that should be retrieved. You should either specify this parameter or `packageNames`.

**packageNames**

The name of the package or packages that should be retrieved.

**Table 1: build.xml retrieve target field settings**

| Field | Description |
|---|---|
| `username` | Required. The Database.com username for login. |
| `password` | Required. The username 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. The username associated with this connection must have the "Modify All Data" permission. Typically, this is only enabled for System Administrator users. |
| `serverurl` | Optional. The salesforce server URL (if blank, defaults to `www.salesforce.com`). For a test database, use `test.salesforce.com`. |

| Field | Description |
|---|---|
| `pollWaitMillis` | Optional, defaults to 5000. The number of milliseconds to wait between each poll of salesforce.com to retrieve the results. |
| `maxPoll` | Optional, defaults to 10. The number of times to poll salesforce.com for the results of the report. |
| `retrieveTarget` | Required. The root of the directory structure to retrieve the metadata files into. |
| `unpackaged` | Optional. The name of a file manifest that specifies the components to retrieve. |
| `singlePackage` | Optional, defaults to false. Specifies whether the contents being retrieved are a single package. |
| `packageNames` | Optional. A list of the names of the packages to retrieve. |
| `specificFiles` | Optional. A list of file names to retrieve. |

### Understanding `runTests()`

In addition to using `deploy()` with the Force.com Migration Tool, you can also use the `runTests()` API call. This call takes the following properties:

**`class`**

The name of the class that contains the unit tests. You can specify this property more than once.

**`alltests`**

Specifies whether to run all tests. This property takes a Boolean value: `true` if you want to run all tests, `false` otherwise.

**`namespace`**

The namespace that you would like to test. If you specify a namespace, all the tests in that namespace are executed.

## 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 test database 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

# 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 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 1073 in the Appendices section.

### *Auth Namespace*

The `Auth` namespace provides an interface and classes for single sign-on into Database.com.

### *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.

### *Database Namespace*

The `Database` namespace provides classes used with DML operations.

### *Dom Namespace*

The `Dom` namespace provides classes and methods for approval processing.

### *QuickAction Namespace*

The `QuickAction` namespace provides classes and methods for publisher actions.

### *Schema Namespace*

The `Schema` namespace provides classes and methods for schema metadata information.

### *System Namespace*

The `System` namespace provides classes and methods for core Apex functionality.

# DML Operations

You can perform DML operations using the DML statements or the methods of the `Database` class.

# DML Statements

Use Data Manipulation Language (DML) operations to insert, update, merge, delete, and restore data in a database.

The following Apex DML statements are available:



## Insert Statement

The `insert` DML operation adds one or more sObjects 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 invoice statement:

```
Invoice_Statement__c invoice = new Invoice_Statement__c(
   Description__c = 'Invoice 1');
try {
   insert invoice;
} catch (DmlException e) {
// Process exception here
}
```

**Note:** For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 327.

## Update Statement

The `update` DML operation modifies one or more existing sObject records, such as individual invoice 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 `Description__c` field on a single invoice statement:

```
Invoice_Statement__c inv = new Invoice_Statement__c(
                               Description__c='Invoice 1');
insert inv;

Invoice_Statement__c myInvoice = [SELECT Id, Description__c
                                  FROM Invoice_Statement__c
                                  WHERE Id = :inv.Id];
myInvoice.Description__c = 'New description';

try {
    update myInvoice;
} catch (DmlException e) {
    // Process exception here
}
```

**Note:** For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 327.

## Upsert Statement

The `upsert` DML operation creates new sObject records and updates existing sObject records within a single statement, using an optional custom field to determine the presence of existing objects.

### Syntax

```
upsert sObject opt_external_id

upsert sObject[] opt_external_id
```

***opt_external_id*** is an optional variable that specifies the custom field that should be used to match records that already exist in your organization's data. This custom field must be created with the `External Id` attribute selected. Additionally, if the field does not have the `Unique` attribute selected, 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.

If ***opt_external_id*** is not specified, the sObject record's ID field is used by default.

**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 "Creating Custom Fields" in the Database.com online help.

### How Upsert Chooses to Insert or Update

Upsert uses the sObject record's primary key (or the external ID, if specified) to determine whether it should create a new object record or update an existing one:

- 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.

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 Database.com*.

### Example

This example performs an upsert of a list of merchandise items.

```
List<Merchandise__c> merList = new List<Merchandise__c>();
// Fill the list with some merchandise items

try {
    upsert merList;
} catch (DmlException e) {

}
```

This next example performs an upsert of a list of merchandise items using a foreign key for matching existing records, if any.

```
List<Merchandise__c> merList = new List<Merchandise__c>();
// Fill the list with some merchandise items

try {
    // Upsert using an external ID field
    upsert merList myExtIDField__c;
} catch (DmlException e) {

}
```

## Delete Statement

The `delete` DML operation deletes one or more existing sObject records 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 merchandise items that are named 'Pencil':

```
Merchandise__c[] pencils = [SELECT Id, Name FROM Merchandise__c
                            WHERE Name = 'Pencil'];
try {
    delete pencils;
} catch (DmlException e) {
    // Process exception here
}
```

**Note:** For more information on processing `DmlExceptions`, see Bulk DML Exception Handling on page 327.

## Undelete Statement

The `undelete` DML operation restores one or more existing sObject records, such as individual invoice statements. `undelete` is analogous to the UNDELETE statement in SQL.

### Syntax

```
undelete sObject | ID
```

```
undelete sObject[] | ID[]
```

### Example

The following example undeletes an invoice statement. The `ALL ROWS` keyword queries all rows for both top level and aggregate relationships, including deleted records and archived activities.

```
Invoice_Statement__c[] savedInvoices =
                       [SELECT Id
                        FROM Invoice_Statement__c
                        WHERE Description__c = 'My invoice' 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 327.

## 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 `all_or_none` 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, 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 which 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 did not 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:** During the second and third attempts, governor limits are reset to their original state before the first attempt. See Understanding Execution Governors and Limits on page 203.

# Auth Namespace

The `Auth` namespace provides an interface and classes for single sign-on into Database.com.

The following is the interface in the `Auth` namespace.

***AuthToken Class***

Contains methods for providing the access token associated with an authentication provider for an authenticated user, except for the Janrain provider.

***RegistrationHandler Interface***

Database.com provides the ability to use an authentication provider, such as Facebook© or Janrain©, for single sign-on into Database.com.

***UserData Class***

Stores user information for `Auth.RegistrationHandler`.

# 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 instance methods.

***getAccessToken(String, String)***

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 Database.com or Facebook.

***getAccessTokenMap(String, String)***

Returns a map from the third-party identifier to the access token for the currently logged-in Database.com user. The identifier value depends on the third party. For example, for Database.com it would be the user ID, while for Facebook it would be the user number.

***refreshAccessToken(String, String, String)***

Returns a map from the third-party identifier containing a refreshed access token for the currently logged-in Database.com user.

## getAccessToken(String, String)

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 Database.com or Facebook.

### Signature

```
public String getAccessToken(String authProviderId, String providerName)
```

### Parameters

***authProviderId***

Type: String

***providerName***

Type: String

#### Return Value

Type: String

### getAccessTokenMap(String, String)

Returns a map from the third-party identifier to the access token for the currently logged-in Database.com user. The identifier value depends on the third party. For example, for Database.com it would be the user ID, while for Facebook it would be the user number.

#### Signature

```
public Map<String, String> getAccessTokenMap(String authProviderId, String providerName)
```

#### Parameters

***authProviderId***

Type: String

***providerName***

Type: String

#### Return Value

Type: Map<String, String>

### refreshAccessToken(String, String, String)

Returns a map from the third-party identifier containing a refreshed access token for the currently logged-in Database.com user.

#### Signature

```
public 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 Database.com 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()`.

### 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...)
```

# RegistrationHandler Interface

Database.com provides the ability to use an authentication provider, such as Facebook© or Janrain©, for single sign-on into Database.com.

## 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 Database.com 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.



# RegistrationHandler Methods

The following are methods for `RegistrationHandler`.

### *createUser(ID, Auth.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.

### *updateUser(ID, ID, Auth.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(ID, Auth.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(ID, ID, Auth.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 Database.com 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);
}
}
```

# UserData Class

Stores user information for `Auth.RegistrationHandler`.

## Namespace

Auth

***UserData Constructors***
***UserData Properties***

# UserData Constructors

The following are constructors for `UserData`.

***UserData(String, String, String, String, String, String, String, String, String, String, MAP<String,String>)***

Creates a new instance of the `Auth.UserData` class using the specified arguments.

## UserData(String, String, String, String, String, String, String, String, String, String, MAP<String,String>)

Creates a new instance of the `Auth.UserData` class using the specified arguments.

### Signature

```
public 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)
```

### Parameters

***identifier***

Type: String

An identifier from the third party for the authenticated user, such as the Facebook user number or the Database.com 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`.

### *identifier*

An identifier from the third party for the authenticated user, such as the Facebook user number or the Database.com 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 Database.com 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>

# 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 Working with Chatter in Apex on page 211.

### Chatter Class

Access information about followers and subscriptions for records.

### ChatterFavorites Class

Chatter favorites give you easy access to topics, list views, and feed searches.

### ChatterFeeds Class

Get, post, and delete feed items, likes, comments, and bookmarks. You can also search feed items, share feed items, 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.

### 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.

### Organization Class

Access information about an organization.

### 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.

### RecordDetails Class

Access information about records in your organization.

### Records Class

Access information about record motifs, which are small icons used to distinguish record types in the salesforce.com 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, and add and remove topics from records and feed items.

### *UserProfiles Class*

Access user profile data. 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 `ConnectApi`.

### *ConnectApi Exceptions*

The `ConnectApi` namespace contains exception classes.

# Chatter Class

Access information about followers and subscriptions for records.

## Namespace

ConnectApi

## Chatter Methods

The following are methods for `Chatter`. All methods are static.

### *deleteSubscription(String, String)*

Deletes the specified subscription. Use this method to unfollow a record, a user, or a file.

### *getFollowers(String, String)*

Returns the first page of followers for the specified record in the specified community. The page contains the default number of items.

### *getFollowers(String, String, Integer, Integer)*

Returns the specified page of followers for the specified record.

### *getSubscription(String, String)*

Returns information about the specified subscription.

## deleteSubscription(String, String)

Deletes the specified subscription. Use this method to unfollow a record, a user, or a file.

#### API Version

28.0

#### Signature

```
public static void deleteSubscription(String communityId, String subscriptionId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***subscriptionId***

Type: String

The ID for a subscription.

#### Return Value

Type: void

#### Usage

To leave a group, call deleteMember(String, String) on page 460.

### getFollowers(String, String)

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

#### Signature

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String recordId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***recordId***

Type: String

The ID for a record or the keyword `me`.

#### Return Value

Type: `ConnectApi.FollowerPage Class`

### getFollowers(String, String, Integer, Integer)

Returns the specified page of followers for the specified record.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String recordId,
Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.FollowerPage Class`

### getSubscription(String, String)

Returns information about the specified subscription.

#### API Version

28.0

#### Signature

```
public static ConnectApi.Subscription getSubscription(String communityId, String
subscriptionId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***subscriptionId***

Type: String

The ID for a subscription.

#### Return Value

Type: `ConnectApi.Subscription Class`

# 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.

***ChatterFavorites Methods***

# ChatterFavorites Methods

The following are methods for `ChatterFavorites`. All methods are static.

***addFavorite(String, String, String)***

Adds a feed search favorite for the specified user in the specified community.

***addRecordFavorite(String, String, String)***

Adds a topic as a favorite.

***deleteFavorite(String, String, String)***

Deletes the specified favorite.

***getFavorite(String, String, String)***

Returns a description of the favorite.

***getFavorites(String, String)***

Returns a list of all favorites for the specified user in the specified community.

***getFeedItems(String, String, String)***

Returns the first page of feed items for the specific favorite in the specified community. The page contains the default number of items.

***getFeedItems(String, String, String, String, Integer, ConnectApi.FeedSortOrder)***

Returns the specified page of feed items for the specified favorite, in the specified community in the specified order.

***getFeedItems(String, String, String, Integer, String, Integer, FeedSortOrder)***

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.

***setTestGetFeedItems(String, String, String, ConnectApi.FeedItemPage)***

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItems` is called with matching parameters in a test context. You must use the method with the same parameters or the code throws an exception.

#### *setTestGetFeedItems(String, String, String, String, Integer, FeedSortOrder, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItems` is called with matching parameters in a test context. You must use the method with the same parameters or the code throws an exception.

#### *setTestGetFeedItems(String, String, String, Integer, String, Integer, FeedSortOrder, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItems` is called with matching parameters in a test context. You must use the method with the same parameters or the code throws an exception.

#### *updateFavorite(String, String, String, Boolean)*

Updates the last view date of the saved search or list view feed to the current system time if you specify `true` for *`updateLastViewDate`*.

## addFavorite(String, String, String)

Adds a feed search favorite for the specified user in the specified community.

### API Version

28.0

### Signature

```
public static ConnectApi.FeedFavorite addFavorite(String communityId, String subjectId,
String searchText)
```

### Parameters

***communityId***

Type: String

Use `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(String, String, String)

Adds a topic as a favorite.

### API Version

28.0

#### Signature

```
public static ConnectApi.FeedFavorite addRecordFavorite(String communityId, String subjectId,
String targetId)
```

#### Parameters

***communityId***

Type: String

Use `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(String, String, String)

Deletes the specified favorite.

#### API Version

28.0

#### Signature

```
public static Void deleteFavorite(String communityId, String subjectId, String favoriteId)
```

#### Parameters

***communityId***

Type: String

Use `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(String, String, String)

Returns a description of the favorite.

### API Version

28.0

### Signature

```
public static ConnectApi.FeedFavorite getFavorite(String communityId, String subjectId,
String favoriteId)
```

### Parameters

***communityId***

Type: String

Use `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(String, String)

Returns a list of all favorites for the specified user in the specified community.

### API Version

28.0

### Signature

```
public static ConnectApi.FeedFavorites getFavorites(String communityId, String subjectId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***subjectId***

Type: String

Specify the ID for the context user or the alias `me`.

#### Return Value

Type: `ConnectApi.FeedFavorites`

### getFeedItems(String, String, String)

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

#### Signature

```
public static ConnectApi.FeedItemPage getFeedItems(String communityId, String subjectId,
String favoriteId)
```

#### Parameters

***communityId***

Type: String

Use `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`). You must use the set test method with the same parameters or the code throws an exception.

**See Also:**

*setTestGetFeedItems(String, String, String, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

### getFeedItems(String, String, String, String, Integer, ConnectApi.FeedSortOrder)

Returns the specified page of feed items for the specified favorite, in the specified community in the specified order.

#### API Version

28.0

### Signature

```
public static ConnectApi.FeedItemPage getFeedItems(String communityId, String subjectId,
String favoriteId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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`). You must use the set test method with the same parameters or the code throws an exception.

## See Also:

*setTestGetFeedItems(String, String, String, String, Integer, FeedSortOrder, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## getFeedItems(String, String, String, Integer, String, Integer, FeedSortOrder)

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

### 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: FeedSortOrder

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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`). You must use the set test method with the same parameters or the code throws an exception.

**See Also:**

*setTestGetFeedItems(String, String, String, Integer, String, Integer, FeedSortOrder, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

### setTestGetFeedItems(String, String, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItems` 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

28.0

#### Signature

```
public static Void setTestGetFeedItems(String communityId, String subjectId, String
favoriteId, ConnectApi.FeedItemPage result)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

The object containing test data.

#### Return Value

Type: Void

**See Also:**

*getFeedItems(String, String, String)*
*Testing ConnectApi Code*

## setTestGetFeedItems(String, String, String, String, Integer, FeedSortOrder, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItems` 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

28.0

### 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: FeedSortOrder

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItems(String, String, String, String, Integer, ConnectApi.FeedSortOrder)*
*Testing ConnectApi Code*

## setTestGetFeedItems(String, String, String, Integer, String, Integer, FeedSortOrder, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItems` 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

29.0

#### 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: FeedSortOrder

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItems(String, String, String, Integer, String, Integer, FeedSortOrder)*
*Testing ConnectApi Code*

## updateFavorite(String, String, String, Boolean)

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

#### Signature

```
public static ConnectApi.FeedFavorite updateFavorite(String communityId, String subjectId,
String favoriteId, Boolean updateLastViewDate)
```

#### Parameters

***communityId***

Type: String

Use `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`

# ChatterFeeds Class

Get, post, and delete feed items, likes, comments, and bookmarks. You can also search feed items, share feed items, and vote on polls.

## Namespace

ConnectApi

## Usage

Feeds are made up of feed items. A feed item is a piece of information posted by a user (for example, a poll) or by an automated process (for example, when a tracked field is updated on a record). Because feeds and feed items are the core of Chatter, understanding them is crucial to developing applications with Chatter REST API and Chatter in Apex. For detailed information about the composition of feeds and feed items, see Working with Feeds and Feed Items.

Message segments in a feed item are typed as `ConnectApi.MessageSegment`. Feed item attachments are typed as `ConnectApi.FeedItemAttachment`. 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.

***deleteComment(String, String)***

Deletes the specified comment. You can find a comment ID in any feed, such as a news feed or a record feed.

***deleteFeedItem(String, String)***

Deletes the specified feed item.

***deleteLike(String, String)***

Deletes the specified like. This can be a like on a comment or a feed item.

***getComment(String, String)***

Returns the specified comment.

***getCommentsForFeedItem(String, String)***

Returns the first page of comments for the specified feed item. The page contains the default number of items.

### *getCommentsForFeedItem(String, String, String, Integer)*

Returns the specified page of comments for the specified feed item.

### *getFeed(String, ConnectApi.FeedType)*

Returns information about the feed for the specified feed type.

### *getFeed(String, ConnectApi.FeedType, ConnectApi.FeedSortOrder)*

Returns the feed for the specified feed type in the specified order.

### *getFeed(String, ConnectApi.FeedType, String)*

Returns the feed for the specified feed type for the specified user.

### *getFeed(String, ConnectApi.FeedType, String, ConnectApi.FeedSortOrder)*

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 logged-in user.

### *getFeedItem(String, String)*

Returns a rich description of the specified feed item.

### *getFeedItemsFromFeed(String, ConnectApi.FeedType)*

Returns the first page of feed items for the `Company` feed type. The page contains the default number of items.

### *getFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder)*

Returns the feed items for the specified page for the `Company` feed type, in the specified order.

### *getFeedItemsFromFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder)*

Returns the feed items for the specified page for the `Company` feed type, in the specified order. Each feed item contains no more than the specified number of comments.

### *getFeedItemsFromFeed(String, ConnectApi.FeedType, String)*

Returns the first page of feed items for the specified feed type, for the specified user. The page contains the default number of items.

### *getFeedItemsFromFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder)*

Returns the feed items on the specified page for the specified user, for the specified feed type in the specified order.

### *getFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder)*

Returns the feed items on the specified page for the specified user, for the specified feed type in the specified order. Each feed item includes no more than the specified number of comments.

### *getFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, Boolean)*

Returns the feed items on the specified page for the specified user, for the specified 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(String, String, String)*

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(String, String, String, String, Integer, ConnectApi.FeedSortOrder)*

Returns the specified page of feed items for the specified user and the specified key prefix, in the specified order.

### *getFeedItemsFromFilterFeed(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder)*

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(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, String)*

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(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, String)*

Returns the specified page of feed items for the `Company` feed type. 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(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String)*

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(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String, Boolean)*

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(String, String)*

Returns the poll associated with the specified feed item.

### *getFilterFeed(String, String, String)*

Returns the first page of a feed for the specified user and the given key prefix.

### *getFilterFeed(String, String, String, ConnectApi.FeedType)*

Returns the first page of a feed in the specified order for the specified user and the given key prefix.

### *getFilterFeedDirectory(String, String)*

Gets a feed directory object that contains a list of filter feeds available to the logged-in user. A filter feed is the news feed filtered to include feed items whose parent is a specific entity type.

### *getLike(String, String)*

Returns the specified like.

### *getLikesForComment(String, String)*

Returns the first page of likes for the specified comment. The page contains the default number of items.

### *getLikesForComment(String, String, Integer, Integer)*

Returns the specified page of likes for the specified comment.

### *getLikesForFeedItem(String, String)*

Returns the first page of likes for the specified feed item. The page contains the default number of items.

### *getLikesForFeedItem(String, String, Integer, Integer)*

Returns the specified page of likes for the specified feed item.

### *isModified(String, ConnectApi.FeedType, String, String)*

Returns information about whether a news feed has been updated or changed. Use this method to poll a news feed for updates.

### *likeComment(String, String)*

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.

### *likeFeedItem(String, String)*

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(String, String, String)*

Adds the specified text as a comment to the specified feed item, for the context user.

### *postComment(String, String, ConnectApi.CommentInput, ConnectApi.BinaryInput)*

Adds a comment to the specified feed item from the context user. Use this method to use rich text, including mentions, and to attach a file to a comment.

### *postFeedItem(String, ConnectApi.FeedType, String, String)*

Adds a feed item to the specified feed from the context user.

### *postFeedItem(String, ConnectApi.FeedType, String, ConnectApi.FeedItemInput, ConnectApi.BinaryInput)*

Adds a feed item to the specified feed from the context user Use this method to post rich text, including @mentions and hashtags, and to attach a file to a feed item. You can also use this method to share a feed item and add a comment.

### *searchFeedItems(String, String)*

Returns the first page of all the feed items that match the specified search criteria. The page contains the default number of items.

### *searchFeedItems(String, String, ConnectApi.FeedSortOrder)*

Returns the first page of all the feed items that match the specified search criteria. The page contains the default number of items.

### *searchFeedItems(String, String, String, Integer)*

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

### *searchFeedItems(String, String, String, Integer, ConnectApi.FeedSortOrder)*

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

### *searchFeedItems(String, String, Integer, String, Integer, ConnectApi.FeedSortOrder)*

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

### *searchFeedItemsInFeed(String, ConnectApi.FeedType, String)*

Searches the feed items for the `Company` feed type.

### *searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder, String)*

Searches the feed items for the `Company` feed type and returns a specified page and page size in a specified sort order.

### *searchFeedItemsInFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String)*

Searches the feed items for the `Company` 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, String)*

Searches the feed items for a specified feed type.

### *searchFeedItemsInFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder, String)*

Searches the feed items for a specified feed type and context user, and returns a specified page and page size in a specified sort order.

### *searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String)*

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 context user, 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(String, String, String, String)*

Searches the feed items of a feed filtered by key prefix.

### *searchFeedItemsInFilterFeed(String, String, String, String, Integer, ConnectApi.FeedSortOrder, String)*

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(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String)*

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.

### *setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, Boolean, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFilterFeed(String, String, String, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `getFeedItemsFromFilterFeed` method is called in a test context. You must use the method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFilterFeed(String, String, String, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `getFeedItemsFromFilterFeed` method is called in a test context. You must use the method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFilterFeed(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `getFeedItemsFromFilterFeed` method is called in a test context. You must use the method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsFromFilterFeedUpdatedSince(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, String, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when the `getFeedItemsFromFilterFeedUpdatedSince` method is called in a test context.

### *setTestGetFeedItemsUpdatedSince(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, String, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsUpdatedSince` is called with matching parameters in a test context. You must use the method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsUpdatedSince(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsUpdatedSince` is called with matching parameters in a test context. You must use the method with the same parameters or the code throws an exception.

### *setTestGetFeedItemsUpdatedSince(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String, Boolean, ConnectApi.FeedItemPage)*

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsUpdatedSince` is called with matching parameters in a test context. You must use the method with the same parameters or the code throws an exception.

### *setTestSearchFeedItems(String, String, ConnectApi.FeedItemPage)*

Registers a test feed item page to be returned when `searchFeedItems(String, String)` is called during a test.

### setTestSearchFeedItems(String, String, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a test feed item page to be returned when `searchFeedItems(String, String, ConnectApi.FeedSortOrder)` is called during a test.

### setTestSearchFeedItems(String, String, String, Integer, ConnectApi.FeedItemPage)

Registers a test feed item page to be returned when `searchFeedItems(String, String, String, Integer)` is called during a test.

### setTestSearchFeedItems(String, String, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a test feed item page to be returned when `searchFeedItems(String, String, String, Integer, ConnectApi.FeedSortOrder)` is called during a test.

### setTestSearchFeedItems(String, String, Integer, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a test feed item page to be returned when `searchFeedItems(String, String, Integer, String, Integer, ConnectApi.FeedSortOrder)` is called during a test.

### setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

### setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

### setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

### setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

### setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

### setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

***setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, Boolean, ConnectApi.FeedItemPage)***

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

***setTestSearchFeedItemsInFilterFeed(String, String, String, String, ConnectApi.FeedItemPage)***

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFilterFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

***setTestSearchFeedItemsInFilterFeed(String, ConnectApi.FeedType, String, String, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)***

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFilterFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

***setTestSearchFeedItemsInFilterFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)***

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFilterFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

***shareFeedItem(String, ConnectApi.FeedType, String, String)***

Share the *`originalFeedItemId`* to the feed specified by the *`feedType`*.

***updateBookmark(String, String, Boolean)***

Bookmarks the specified feed item or removes a bookmark from the specified feed item.

***voteOnFeedPoll(String, String, String)***

Used to vote or to change your vote on an existing feed poll.

## deleteComment(String, String)

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

### Signature

```
public static Void deleteComment(String communityId, String commentId)
```

### Parameters

***`communityId`***

Type: String

Use `null`.

***`commentId`***

Type: String

The ID for a comment.

#### Return Value

Type: Void

### deleteFeedItem(String, String)

Deletes the specified feed item.

#### API Version

28.0

#### Signature

```
public static Void deleteFeedItem(String communityId, String feedItemId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

#### Return Value

Type: Void

### deleteLike(String, String)

Deletes the specified like. This can be a like on a comment or a feed item.

#### API Version

28.0

#### Signature

```
public static Void deleteLike(String communityId, String likeId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***likeId***

Type: String

The ID for a like.

#### Return Value

Type: Void

### getComment(String, String)

Returns the specified comment.

#### API Version

28.0

#### Signature

```
public static ConnectApi.Comment getComment(String communityId, String commentId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***commentId***

Type: String

The ID for a comment.

#### Return Value

Type: `ConnectApi.Comment Class`

### getCommentsForFeedItem(String, String)

Returns the first page of comments for the specified feed item. The page contains the default number of items.

#### API Version

28.0

#### Signature

```
public static ConnectApi.CommentPage getCommentsForFeedItem(String communityId, String
feedItemId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

#### Return Value

Type: ConnectApi.CommentPage Class

### getCommentsForFeedItem(String, String, String, Integer)

Returns the specified page of comments for the specified feed item.

#### API Version

28.0

#### Signature

```
public static ConnectApi.CommentPage getCommentsForFeedItem(String communityId, String feedItemId, String pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use 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 items per page. Valid values are between 1 and 100. If you pass in null, the default size is 25.

#### Return Value

Type: ConnectApi.CommentPage Class

### getFeed(String, ConnectApi.FeedType)

Returns information about the feed for the specified feed type.

#### API Version

28.0

#### Signature

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

#### Return Value

Type: `ConnectApi.Feed Class`

### getFeed(String, ConnectApi.FeedType, ConnectApi.FeedSortOrder)

Returns the feed for the specified feed type in the specified order.

#### API Version

28.0

#### Signature

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType,
ConnectApi.FeedSortOrder sortParam)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

### getFeed(String, ConnectApi.FeedType, String)

Returns the feed for the specified feed type for the specified user.

#### API Version

28.0

#### Signature

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType,
String subjectId)
```

#### Parameters

***communityId***

Type: String

Use `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`.

#### Return Value

Type: `ConnectApi.Feed Class`

### getFeed(String, ConnectApi.FeedType, String, ConnectApi.FeedSortOrder)

Returns the feed for the specified feed type for the specified user, in the specified order.

#### API Version

28.0

#### Signature

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType,
String subjectId, ConnectApi.FeedSortOrder sortParam)
```

#### Parameters

***communityId***

Type: String

Use `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.

***sortParam***

Type: ConnectApi.FeedSortOrder

Values are:

- CreatedDateDesc—Sorts the feed items by most recent post date.
- LastModifiedDateDesc—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class

### getFeedDirectory(String)

Returns a list of all feeds available to the logged-in user.

#### API Version

30.0

#### Signature

```
public static ConnectApi.FeedDirectory getFeedDirectory(String communityId)
```

#### Parameters

***communityId***

Type: String

Use null.

#### Return Value

Type: ConnectApi.FeedDirectory

### getFeedItem(String, String)

Returns a rich description of the specified feed item.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItem getFeedItem(String communityId, String feedItemId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

#### Return Value

Type: `ConnectApi.FeedItem Class`

**Note:** Triggers on FeedItem objects run before their attachment information is saved, which means that `ConnectApi.FeedItem.attachment` information may not be available in the trigger.

### getFeedItemsFromFeed(String, ConnectApi.FeedType)

Returns the first page of feed items for the `Company` feed type. The page contains the default number of items.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

#### Return Value

Type: `ConnectApi.FeedItemPage Class`

#### Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. You must use the test method with the same parameters, or you receive an exception.

## See Also:

*setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## getFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder)

Returns the feed items for the specified page for the `Company` feed type, in the specified order.

### API Version

28.0

### Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder
sortParam)
```

### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value 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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

### Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. You must use the test method with the same parameters, or you receive an exception.

### See Also:

*setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedItemsFromFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder)

Returns the feed items for the specified page for the `Company` feed type, in the specified order. Each feed item contains no more than the specified number of comments.

### API Version

29.0

### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

#### Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. You must use the test method with the same parameters, or you receive an exception.

### See Also:

*setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedItemsFromFeed(String, ConnectApi.FeedType, String)

Returns the first page of feed items for the specified feed type, for the specified user. The page contains the default number of items.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId)
```

#### Parameters

***communityId***

Type: String

Use `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`.

#### Return Value

Type: `ConnectApi.FeedItemPage Class`

#### Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. You must use the test method with the same parameters, or you receive an exception.

### See Also:

*setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## getFeedItemsFromFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder)

Returns the feed items on the specified page for the specified user, for the specified feed type in the specified order.

#### API Version

28.0

#### 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 `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

The page contains the default number of items.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

#### Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. You must use the test method with the same parameters, or you receive an exception.

### See Also:

*setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder)

Returns the feed items on the specified page for the specified user, 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

#### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

### Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. You must use the test method with the same parameters, or you receive an exception.

### See Also:

*setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, Boolean)

Returns the feed items on the specified page for the specified user, for the specified 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

### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.

- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

#### Usage

When invoked during a test with the matching `setTestGetFeedItemsFromFeed` method, returns the feed item page passed in with that method. You must use the test method with the same parameters, or you receive an exception.

### See Also:

*setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, Boolean, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedItemsFromFilterFeed(String, String, String)

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

#### Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeed(String communityId, String subjectId, String keyPrefix)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

#### Usage

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:

*setTestGetFeedItemsFromFilterFeed(String, String, String, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## getFeedItemsFromFilterFeed(String, String, String, String, Integer, ConnectApi.FeedSortOrder)

Returns the specified page of feed items for the specified user and the specified key prefix, in the specified order.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

#### Usage

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:

*setTestGetFeedItemsFromFilterFeed(String, String, String, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedItemsFromFilterFeed(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder)

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

### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

#### Usage

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:

*setTestGetFeedItemsFromFilterFeed(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedItemsFromFilterFeedUpdatedSince(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, String)

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

#### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 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(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, String, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## getFeedItemsUpdatedSince(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, String)

Returns the specified page of feed items for the `Company` feed type. 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

### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 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 Class`

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(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, String, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedItemsUpdatedSince(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String)

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

### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 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 Class`

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(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedItemsUpdatedSince(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String, Boolean)

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

### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 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 Class`

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(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String, Boolean, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## getFeedPoll(String, String)

Returns the poll associated with the specified feed item.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedPoll getFeedPoll(String communityId, String feedItemId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

#### Return Value

Type: `ConnectApi.FeedPoll Class`

**Note:** Triggers on FeedItem objects run before their attachment information is saved, which means that `ConnectApi.FeedItem.attachment` information may not be available in the trigger.

## getFilterFeed(String, String, String)

Returns the first page of a feed for the specified user and the given key prefix.

### API Version

28.0

### Signature

```
public static ConnectApi.Feed getFilterFeed(String communityId, String subjectId, String keyPrefix)
```

### Parameters

***communityId***

Type: String

Use `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 Class`

## getFilterFeed(String, String, String, ConnectApi.FeedType)

Returns the first page of a feed in the specified order for the specified user and the given key prefix.

### API Version

28.0

### Signature

```
public static ConnectApi.Feed getFilterFeed(String communityId, String subjectId, String keyPrefix, ConnectApi.FeedType sortParam)
```

### Parameters

***communityId***

Type: String

Use `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 the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

### getFilterFeedDirectory(String, String)

Gets a feed directory object that contains a list of filter feeds available to the logged-in user. A filter feed is the news feed filtered to include feed items whose parent is a specific entity type.

#### API Version

30.0

#### Signature

```
public static ConnectApi.FeedDirectory getFilterFeedDirectory(String communityId, String subjectId)
```

#### Parameters

***communityId***

Type: String

Use `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 logged-in 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 logged-in 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 logged-in 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(String, String)

Returns the specified like.

### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterLike getLike(String communityId, String likeId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***likeId***

Type: String

The ID for a like.

#### Return Value

Type: `ConnectApi.ChatterLike Class`

### getLikesForComment(String, String)

Returns the first page of likes for the specified comment. The page contains the default number of items.

#### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterLikePage getLikesForComment(String communityId, String
commentId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***commentId***

Type: String

The ID for a comment.

#### Return Value

Type: `ConnectApi.ChatterLikePage Class`

### getLikesForComment(String, String, Integer, Integer)

Returns the specified page of likes for the specified comment.

#### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterLikePage getLikesForComment(String communityId, String commentId, Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.ChatterLikePage Class`

### getLikesForFeedItem(String, String)

Returns the first page of likes for the specified feed item. The page contains the default number of items.

#### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterLikePage getLikesForFeedItem(String communityId, String feedItemId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

#### Return Value

Type: `ConnectApi.ChatterLikePage Class`

## getLikesForFeedItem(String, String, Integer, Integer)

Returns the specified page of likes for the specified feed item.

### API Version

28.0

### Signature

```
public static ConnectApi.ChatterLikePage getLikesForFeedItem(String communityId, String feedItemId, Integer pageParam, Integer pageSize)
```

### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

### Return Value

Type: `ConnectApi.ChatterLikePage Class`

## isModified(String, ConnectApi.FeedType, String, String)

Returns information about whether a news feed has been updated or changed. Use this method to poll a news feed for updates.

### API Version

28.0

### Signature

```
public static ConnectApi.FeedModifiedInfo isModified(String communityId, ConnectApi.FeedType feedType, String subjectId, String since)
```

### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: ConnectApi.FeedType

Specifies the type of feed. The only supported type is `News`

***subjectId***

Type: String

The ID of the context user or the alias `me`.

***since***

Type: String

an opaque token containing information about the last modified date of the feed. Retrieve this token from `Feed.isModifiedToken` or `FeedItemPage.isModifiedToken`.

#### Return Value

Type: `ConnectApi.FeedModifiedInfo Class`

#### Usage

For more information, see News Feed Is-Modified in *Chatter REST API Developer's Guide*.

### likeComment(String, String)

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

#### Signature

```
public static ConnectApi.ChatterLike likeComment(String communityId, String commentId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***commentId***

Type: String

The ID for a comment.

#### Return Value

Type: `ConnectApi.ChatterLike Class`

### likeFeedItem(String, String)

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

#### Signature

```
public static ConnectApi.ChatterLike likeFeedItem(String communityId, String feedItemId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

#### Return Value

Type: `ConnectApi.ChatterLike Class`

### postComment(String, String, String)

Adds the specified text as a comment to the specified feed item, for the context user.

#### API Version

28.0

#### Signature

```
public static ConnectApi.Comment postComment(String communityId, String feedItemId, String
text)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

***text***

Type: String

Specify the text of the post.

#### Return Value

Type: `ConnectApi.Comment Class`

### 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(String, String, ConnectApi.CommentInput, ConnectApi.BinaryInput)

Adds a comment to the specified 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

### Signature

```
public static ConnectApi.Comment postComment(String communityId, String feedItemId,
ConnectApi.CommentInput comment, ConnectApi.BinaryInput feedItemFileUpload)
```

### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

***comment***

Type: `ConnectApi.CommentInput Class`

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 Class`

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 Class`

### 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);
```

## postFeedItem(String, ConnectApi.FeedType, String, String)

Adds a feed item to the specified feed from the context user.

### API Version

28.0

### Signature

```
public static ConnectApi.FeedItem postFeedItem(String communityId, ConnectApi.FeedType
feedType, String subjectId, String text)
```

### Parameters

***communityId***

Type: String

Use `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 comment. Mentions and hashtags are removed.

#### Return Value

Type: `ConnectApi.FeedItem Class`

**Note:** Triggers on FeedItem objects run before their attachment information is saved, which means that `ConnectApi.FeedItem.attachment` 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(String, ConnectApi.FeedType, String, ConnectApi.FeedItemInput, ConnectApi.BinaryInput)

Adds a feed item to the specified feed from the context user Use this method to post rich text, including @mentions and hashtags, 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

#### Signature

```
public static ConnectApi.FeedItem postFeedItem(String communityId, ConnectApi.FeedType
feedType, String subjectId, ConnectApi.FeedItemInput feedItemInput, ConnectApi.BinaryInput
feedItemFileUpload)
```

#### Parameters

***`communityId`***

Type: String

Use `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 Class`

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 Class`

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 Class`

**Note:** Triggers on FeedItem objects run before their attachment information is saved, which means that `ConnectApi.FeedItem.attachment` 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);
```

## searchFeedItems(String, String)

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

### Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q)
```

### Parameters

***communityId***

Type: String

Use `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 Class`

### See Also:

*setTestSearchFeedItems(String, String, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## searchFeedItems(String, String, ConnectApi.FeedSortOrder)

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

### Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q,
ConnectApi.FeedSortOrder sortParam)
```

### Parameters

***communityId***

Type: String

Use `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 the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

### See Also:

*setTestSearchFeedItems(String, String, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## searchFeedItems(String, String, String, Integer)

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q, String pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.FeedItemPage Class`

### See Also:

*setTestSearchFeedItems(String, String, String, Integer, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## searchFeedItems(String, String, String, Integer, ConnectApi.FeedSortOrder)

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

### See Also:

*setTestSearchFeedItems(String, String, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## searchFeedItems(String, String, Integer, String, Integer, ConnectApi.FeedSortOrder)

Returns a list of all the feed items viewable by the context user that match the specified search criteria.

#### API Version

29.0

#### Signature

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q, Integer
recentCommentCount, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

## See Also:

*setTestSearchFeedItems(String, String, Integer, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## searchFeedItemsInFeed(String, ConnectApi.FeedType, String)

Searches the feed items for the `Company` feed type.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String q)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

***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 Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder, String)

Searches the feed items for the `Company` feed type and returns a specified page and page size in a specified sort order.

#### API Version

28.0

#### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value 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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## searchFeedItemsInFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String)

Searches the feed items for the `Company` 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

### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## searchFeedItemsInFeed(String, ConnectApi.FeedType, String, String)

Searches the feed items for a specified feed type.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String q)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of the feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, and `Topics`.

***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 Class`

### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, String, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## searchFeedItemsInFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder, String)

Searches the feed items for a specified feed type and context user, and returns a specified page and page size in a specified sort order.

### API Version

28.0

### 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

The type of the feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, and `Topics`.

***feedType***

Type: `ConnectApi.FeedType`

If *`feedType`* is `Record`, *`subjectId`* can be any record ID, including a group ID. If *`feedType`* 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`.

***subjectId***

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.

***pageParam***

Type: String

Specifies the number of items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***pageSize***

Type: Integer

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Required and cannot be `null`. Specifies the string to search. The search string must contain at least two characters, not including wildcards. See Wildcards.

***q***

Type: String

Search term. Searches keywords in the user or group name. A minimum of 1 character is required. This parameter does not support wildcards. This parameter is required.

#### Return Value

Type: `ConnectApi.FeedItemPage Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String)

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

#### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of the feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, and `Topics`.

***subjectId***

Type: String

If *`feedType`* is `Record`, *`subjectId`* can be any record ID, including a group ID. If *`feedType`* 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`.

***recentCommentCount***

Type: Integer

Specify the amount of content in a feed.

- `AllUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***density***

Type: `ConnectApi.FeedDensity`

Specify the amount of content in a feed.

- `AllUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)*

*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 context user, 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

#### 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

The type of the feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, and `Topics`.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

### Usage

To test code that uses this method, use the matching set test method.

## See Also:

*setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, Boolean, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## searchFeedItemsInFilterFeed(String, String, String, String)

Searches the feed items of a feed filtered by key prefix.

### API Version

28.0

### Signature

```
public static ConnectApi.FeedItemPage searchFeedItemsInFilterFeed(String communityId, String subjectId, String keyPrefix, String q)
```

### Parameters

***communityId***

Type: String

Use `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 Class`

### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestSearchFeedItemsInFilterFeed(String, String, String, String, ConnectApi.FeedItemPage)*
*Testing ConnectApi Code*

## searchFeedItemsInFilterFeed(String, String, String, String, Integer, ConnectApi.FeedSortOrder, String)

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

### 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestSearchFeedItemsInFilterFeed(String, ConnectApi.FeedType, String, String, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## searchFeedItemsInFilterFeed(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String)

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

### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestSearchFeedItemsInFilterFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)*

*Testing ConnectApi Code*

## setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

#### API Version

28.0

#### Signature

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType
feedType, ConnectApi.FeedItemPage result)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

***result***

Type: `ConnectApi.FeedItemPage Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsFromFeed(String, ConnectApi.FeedType)*
*Testing ConnectApi Code*

## setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

#### API Version

28.0

#### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value 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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder)*
*Testing ConnectApi Code*

## setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

#### API Version

29.0

#### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

### Return Value

Type: Void

## See Also:

*getFeedItemsFromFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder)*

*Testing ConnectApi Code*

## setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

### API Version

28.0

### Signature

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, ConnectApi.FeedItemPage result)
```

### Parameters

***communityId***

Type: String

Use `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`.

***result***

Type: `ConnectApi.FeedItemPage Class`

The object containing test data.

### Return Value

Type: Void

### See Also:

*getFeedItemsFromFeed(String, ConnectApi.FeedType, String)*
*Testing ConnectApi Code*

## setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

### API Version

28.0

### 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsFromFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder)*
*Testing ConnectApi Code*

## setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

#### API Version

29.0

#### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.

- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

## See Also:

*getFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder)*
*Testing ConnectApi Code*

## setTestGetFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, Boolean, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsFromFeed` is called with matching parameters in a test context. You must use the get feed method with the same parameters or the code throws an exception.

#### API Version

30.0

#### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsFromFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, Boolean)*
*Testing ConnectApi Code*

## setTestGetFeedItemsFromFilterFeed(String, String, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `getFeedItemsFromFilterFeed` method is called in a test context. You must use the method with the same parameters or the code throws an exception.

#### API Version

28.0

#### Signature

```
public static Void setTestGetFeedItemsFromFilterFeed(String communityId, String subjectId,
String keyPrefix, ConnectApi.FeedItemPage result)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsFromFilterFeed(String, String, String)*
*Testing ConnectApi Code*

## setTestGetFeedItemsFromFilterFeed(String, String, String, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `getFeedItemsFromFilterFeed` method is called in a test context. You must use the method with the same parameters or the code throws an exception.

#### API Version

28.0

#### 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsFromFilterFeed(String, String, String, String, Integer, ConnectApi.FeedSortOrder)*
*Testing ConnectApi Code*

## setTestGetFeedItemsFromFilterFeed(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `getFeedItemsFromFilterFeed` method is called in a test context. You must use the method with the same parameters or the code throws an exception.

#### API Version

29.0

#### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsFromFilterFeed(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder)*

*Testing ConnectApi Code*

## setTestGetFeedItemsFromFilterFeedUpdatedSince(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the `getFeedItemsFromFilterFeedUpdatedSince` method is called in a test context.

### API Version

30.0

### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.

- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsFromFilterFeedUpdatedSince(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, String)*
*Testing ConnectApi Code*

## setTestGetFeedItemsUpdatedSince(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsUpdatedSince` 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

30.0

#### Signature

```
public static Void setTestGetFeedItemsUpdatedSince(String communityId, ConnectApi.FeedType
feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam,
Integer pageSize, String updatedSince)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsUpdatedSince(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, String)*
*Testing ConnectApi Code*

## setTestGetFeedItemsUpdatedSince(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsUpdatedSince` 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

30.0

### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getFeedItemsUpdatedSince(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String)*
*Testing ConnectApi Code*

## setTestGetFeedItemsUpdatedSince(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String, Boolean, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when `getFeedItemsUpdatedSince` 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

30.0

#### 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 `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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 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 Class`

The object containing test data.

### Return Value

Type: Void

## See Also:

*getFeedItemsUpdatedSince(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, String, Boolean)*

*Testing ConnectApi Code*

## setTestSearchFeedItems(String, String, ConnectApi.FeedItemPage)

Registers a test feed item page to be returned when `searchFeedItems(String, String)` is called during a test.

### API Version

28.0

### Signature

```
public static Void searchFeedItems(String communityId, String q, ConnectApi.FeedItemPage
result)
```

### Parameters

***communityId***

Type: String

Use `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(String, String)*
*Testing ConnectApi Code*

## setTestSearchFeedItems(String, String, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a test feed item page to be returned when `searchFeedItems(String, String, ConnectApi.FeedSortOrder)` is called during a test.

### API Version

28.0

### Signature

```
public static Void setTestSearchFeedItems(String communityId, String q,
ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

#### Parameters

***communityId***

Type: String

Use `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 the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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(String, String, ConnectApi.FeedSortOrder)*
*Testing ConnectApi Code*

## setTestSearchFeedItems(String, String, String, Integer, ConnectApi.FeedItemPage)

Registers a test feed item page to be returned when `searchFeedItems(String, String, String, Integer)` is called during a test.

#### API Version

28.0

#### Signature

```
public static Void setTestSearchFeedItems(String communityId, String q, String pageParam,
Integer pageSize, ConnectApi.FeedItemPage result)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***result***

Type: `ConnectApi.FeedItemPage Class`

The test feed item page.

#### Return Value

Type: Void

## See Also:

*searchFeedItems(String, String, String, Integer)*
*Testing ConnectApi Code*

## setTestSearchFeedItems(String, String, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

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

#### Signature

```
public static Void setTestSearchFeedItems(String communityId, String q, String pageParam,
Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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(String, String, String, Integer, ConnectApi.FeedSortOrder)*
*Testing ConnectApi Code*

## setTestSearchFeedItems(String, String, Integer, String, Integer, ConnectApi.FeedSortOrder, ConnectApi.FeedItemPage)

Registers a test feed item page to be returned when `searchFeedItems(String, String, Integer, String, Integer, ConnectApi.FeedSortOrder)` is called during a test.

#### API Version

29.0

#### 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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(String, String, Integer, String, Integer, ConnectApi.FeedSortOrder)*
*Testing ConnectApi Code*

## setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### API Version

28.0

#### Signature

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType
feedType, String q, ConnectApi.FeedItemPage result)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. The only valid value is `Company`.

***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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*searchFeedItemsInFeed(String, ConnectApi.FeedType, String)*
*Testing ConnectApi Code*

## setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### API Version

28.0

#### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

### Return Value

Type: Void

## See Also:

*searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedSortOrder, String)*
*Testing ConnectApi Code*

## setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

### API Version

29.0

### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

## See Also:

*searchFeedItemsInFeed(String, ConnectApi.FeedType, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String)*

*Testing ConnectApi Code*

## setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### API Version

28.0

#### Signature

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType
feedType, String subjectId, String q, ConnectApi.FeedItemPage result)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of the feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, and `Topics`.

***subjectId***

Type: String

If *feedType* is `Record`, *subjectId* can be any record ID, including a group ID. If *feedType* 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.FeedItemPage Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*searchFeedItemsInFeed(String, ConnectApi.FeedType, String, String)*
*Testing ConnectApi Code*

## setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### API Version

28.0

#### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of the feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, and `Topics`.

***subjectId***

Type: String

If *feedType* is Record, *subjectId* can be any record ID, including a group ID. If *feedType* 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.

***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 items per page. Valid values are between 1 and 100. If you pass in null, the default size is 25.

***sortParam***

Type: ConnectApi.FeedSortOrder

Values are:

- CreatedDateDesc—Sorts the feed items by most recent post date.
- LastModifiedDateDesc—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class

The object containing test data.

#### Return Value

Type: Void

### See Also:

*searchFeedItemsInFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedSortOrder, String)*
*Testing ConnectApi Code*

## setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a ConnectApi.FeedItemPage object to be returned when the matching ConnectApi.searchFeedItemsInFeed method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### API Version

29.0

### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of the feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, and `Topics`.

***subjectId***

Type: String

If *`feedType`* is `Record`, *`subjectId`* can be any record ID, including a group ID. If *`feedType`* 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`.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

The object containing test data.

#### Return Value

Type: Void

## See Also:

*searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String)*
*Testing ConnectApi Code*

## setTestSearchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, Boolean, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### API Version

29.0

#### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of the feed. Valid values include every `ConnectApi.FeedType` except `Company`, `Filter`, and `Topics`.

***subjectId***

Type: String

If *feedType* is Record, *subjectId* can be any record ID, including a group ID. If *feedType* 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.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- FewerUpdates—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in null, the default size is 25.

***sortParam***

Type: ConnectApi.FeedSortOrder

Values are:

- CreatedDateDesc—Sorts the feed items by most recent post date.
- LastModifiedDateDesc—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class

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(String, String, String, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFilterFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### API Version

s

28.0

#### Signature

```
public static Void setTestSearchFeedItemsInFilterFeed(String communityId, String subjectId,
String keyPrefix, String q, ConnectApi.FeedItemPage result)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

Specify the test feed item page.

#### Return Value

Type: Void

### See Also:

*searchFeedItemsInFilterFeed(String, String, String, String)*
*Testing ConnectApi Code*

## setTestSearchFeedItemsInFilterFeed(String, ConnectApi.FeedType, String, String, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFilterFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### API Version

28.0

#### 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 `null`.

***feedType***

Type: `ConnectApi.FeedType`

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: `ConnectApi.FeedSortOrder`

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

Specify the test feed item page.

#### Return Value

Type: Void

### See Also:

*searchFeedItemsInFilterFeed(String, String, String, String, Integer, ConnectApi.FeedSortOrder, String)*
*Testing ConnectApi Code*

## setTestSearchFeedItemsInFilterFeed(String, ConnectApi.FeedType, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, ConnectApi.FeedItemPage)

Registers a `ConnectApi.FeedItemPage` object to be returned when the matching `ConnectApi.searchFeedItemsInFilterFeed` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### API Version

29.0

#### 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 `null`.

***feedType***

Type: ConnectApi.FeedType

The type of feed. Valid values include every `ConnectApi.FeedType` except `Company` and `Filter`.

***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 posts and comments from people and records the user follows and groups the user is a member of.
- `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***sortParam***

Type: ConnectApi.FeedSortOrder

Values are:

- `CreatedDateDesc`—Sorts the feed items by most recent post date.
- `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments.

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 Class`

Specify the test feed item page.

#### Return Value

Type: Void

### See Also:

*searchFeedItemsInFilterFeed(String, String, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String)*
*Testing ConnectApi Code*

## shareFeedItem(String, ConnectApi.FeedType, String, String)

Share the *originalFeedItemId* to the feed specified by the *feedType*.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItem shareFeedItem(String communityId, ConnectApi.FeedType
feedType, String subjectId, String originalFeedItemId)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

**Note:** Triggers on FeedItem objects run before their attachment information is saved, which means that `ConnectApi.FeedItem.attachment` 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(String, String, Boolean)

Bookmarks the specified feed item or removes a bookmark from the specified feed item.

#### API Version

28.0

#### Signature

```
public static ConnectApi.FeedItem updateBookmark(String communityId, String feedItemId,
Boolean isBookmarkedByCurrentUser)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### voteOnFeedPoll(String, String, String)

Used to vote or to change your vote on an existing feed poll.

### API Version

28.0

### Signature

```
public static ConnectApi.FeedPoll voteOnFeedPoll(String communityId, String feedItemId,
String myChoiceId)
```

### Parameters

***communityId***

Type: String

Use `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 Class`

**Note:** Triggers on FeedItem objects run before their attachment information is saved, which means that `ConnectApi.FeedItem.attachment` information may not be available in the trigger.

# 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.

***addMember(String, String, String)***

Adds the specified user to the specified group in the specified community as a standard member.

***addMemberWithRole(String, String, String, ConnectApi.GroupMembershipType)***

Adds the specified user with the specified role to the specified group in the specified community.

***createGroup(String, ConnectApi.ChatterGroupInput)***

Creates a group.

### *deleteMember(String, String)*

Deletes the specified group membership.

### *deletePhoto(String, String)*

Deletes the photo of the specified group.

### *getGroup(String, String)*

Returns information about the specified group.

### *getGroupMembershipRequest(String, String)*

Returns information about the specified request to join a private group.

### *getGroupMembershipRequests(String, String)*

Returns information about every request to join the specified private group.

### *getGroupMembershipRequests(String, String, ConnectApi.GroupMembershipRequestStatus)*

Returns information about every request to join the specified private group that has a specified status.

### *getGroups(String)*

Returns the first page of all the groups. The page contains the default number of items.

### *getGroups(String, Integer, Integer)*

Returns the specified page of information about all groups.

### *getGroups(String, Integer, Integer, ConnectApi.GroupArchiveStatus)*

Returns the specified page of information about a set of groups with a specified group archive status.

### *getMember(String, String)*

Returns information about the specified group member.

### *getMembers(String, String)*

Returns the first page of information about all members of the specified group. The page contains the default number of items.

### *getMembers(String, String, Integer, Integer)*

Returns the specified page of information about all members of the specified group.

### *getMyChatterSettings(String, String)*

Returns the context user's Chatter settings for the specified group.

### *getPhoto(String, String)*

Returns information about the photo for the specified group.

### *requestGroupMembership(String, String)*

Requests membership in a private group for the context user.

### *searchGroups(String, String)*

Returns the first page of groups that match the specified search criteria. The page contains the default number of items.

### *searchGroups(String, String, Integer, Integer)*

Returns the specified page of groups that match the specified search criteria.

***searchGroups(String, String, ConnectApi.GroupArchiveStatus, Integer, Integer)***

Returns the specified page of groups that match the specified search criteria and that have the specified archive status.

***setPhoto(String, String, String, Integer)***

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(String, String, ConnectApi.BinaryInput)***

Sets the group photo to the specified blob..

***setPhotoWithAttributes(String, String, ConnectApi.PhotoInput)***

Sets and crops an already uploaded file as the group photo.

***setPhotoWithAttributes(String, String, ConnectApi.PhotoInput, ConnectApi.BinaryInput)***

Sets and crops a binary input as the group photo.

***setTestSearchGroups(String, String, ConnectApi.ChatterGroupPage)***

Registers a `ConnectApi.ChatterGroupPage` object to be returned when the matching `ConnectApi.searchGroups` method is called in a test context. You must use the test method with the same parameters or you receive an exception.

***setTestSearchGroups(String, String, Integer, Integer, ConnectApi.ChatterGroupPage)***

Registers a `ConnectApi.ChatterGroupPage` object to be returned when the matching `ConnectApi.searchGroups` method is called in a test context. You must use the test method with the same parameters or you receive an exception.

***setTestSearchGroups(String, String, ConnectApi.GroupArchiveStatus, Integer, Integer, ConnectApi.ChatterGroupPage)***

Registers a `ConnectApi.ChatterGroupPage` object to be returned when the matching `ConnectApi.searchGroups` method is called in a test context. You must use the test method with the same parameters or you receive an exception.

***updateGroup(String, String, ConnectApi.ChatterGroupInput)***

Update the settings of a group.

***updateGroupMember(String, String, ConnectApi.GroupMembershipType)***

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(String, String, ConnectApi.GroupEmailFrequency)***

Updates the context user’s Chatter settings for the specified group.

***updateRequestStatus(String, String, ConnectApi.GroupMembershipRequestStatus)***

Updates a request to join a private group.

## addMember(String, String, String)

Adds the specified user to the specified group in the specified community as a standard member.

### API Version

28.0

### Signature

```
public static ConnectApi.GroupMember addMember(String communityId, String groupId, String
userId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

***userId***

Type: String

The ID for a user.

#### Return Value

Type: `ConnectApi.GroupMember Class`

### addMemberWithRole(String, String, String, ConnectApi.GroupMembershipType)

Adds the specified user with the specified role to the specified group in the specified community.

#### API Version

29.0

#### Signature

```
public static ConnectApi.GroupMember addMemberWithRole(String communityId, String groupId,
String userId, ConnectApi.GroupMembershipType role)
```

#### Parameters

***communityId***

Type: String

Use `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 Class

### createGroup(String, ConnectApi.ChatterGroupInput)

Creates a group.

#### API Version

29.0

#### Signature

```
public static ConnectApi.ChatterGroupDetail createGroup(String, communityId,
ConnectApi.ChatterGroupInput groupInput)
```

#### Parameters

***communityId***

Type: String,

Use null.

***groupInput***

Type: ConnectApi.ChatterGroupInput Class

The properties of the group.

#### Return Value

Type: ConnectApi.ChatterGroupDetail Class

### deleteMember(String, String)

Deletes the specified group membership.

#### API Version

28.0

#### Signature

```
public static Void deleteMember(String communityId, String membershipId)
```

#### Parameters

***communityId***

Type: String

Use 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(String, String)

Deletes the photo of the specified group.

#### API Version

28.0

#### Signature

```
public static Void deletePhoto(String communityId, String groupId)
```

#### Parameters

***communityId***

Type: String

Use `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.

### getGroup(String, String)

Returns information about the specified group.

#### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterGroupDetail getGroup(String communityId, String groupId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

#### Return Value

Type: `ConnectApi.ChatterGroupDetail Class`

### getGroupMembershipRequest(String, String)

Returns information about the specified request to join a private group.

#### API Version

28.0

#### Signature

```
public static ConnectApi.GroupMembershipRequest getGroupMembershipRequest(String communityId,
String requestId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***requestId***

Type: String

The ID of a request to join a private group.

#### Return Value

Type: `ConnectApi.GroupMembershipRequest Class`

#### Usage

This method is only successful when the context user is the group manager or owner, or has “Modify All Data” permission.

### getGroupMembershipRequests(String, String)

Returns information about every request to join the specified private group.

#### API Version

28.0

#### Signature

```
public static ConnectApi.GroupMembershipRequests getGroupMembershipRequests(String
communityId, String groupId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

#### Return Value

Type: `ConnectApi.GroupMembershipRequests Class`

#### Usage

This method is only successful when the context user is the group manager or owner, or has “Modify All Data” permission.

### getGroupMembershipRequests(String, String, ConnectApi.GroupMembershipRequestStatus)

Returns information about every request to join the specified private group that has a specified status.

#### API Version

28.0

#### Signature

```
public static ConnectApi.GroupMembershipRequests getGroupMembershipRequests(String
communityId, String groupId, ConnectApi.GroupMembershipRequestStatus status)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

#### Usage

This method is only successful when the context user is the group manager or owner, or has “Modify All Data” permission.

### getGroups(String)

Returns the first page of all the groups. The page contains the default number of items.

#### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterGroupPage getGroups(String communityId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

#### Return Value

Type: `ConnectApi.ChatterGroupPage Class`

### getGroups(String, Integer, Integer)

Returns the specified page of information about all groups.

#### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterGroupPage getGroups(String communityId, Integer pageParam,
Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.ChatterGroupPage Class`

### getGroups(String, Integer, Integer, ConnectApi.GroupArchiveStatus)

Returns the specified page of information about a set of groups with a specified group archive status.

#### API Version

29.0

#### Signature

```
public static ConnectApi.ChatterGroupPage getGroups(String communityId, Integer pageParam,
Integer pageSize, ConnectApi.GroupArchiveStatus archiveStatus)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 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 Class`

### getMember(String, String)

Returns information about the specified group member.

#### API Version

28.0

#### Signature

```
public static ConnectApi.GroupMember getMember(String communityId, String membershipId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***membershipId***

Type: String

The ID for a membership.

#### Return Value

Type: `ConnectApi.GroupMember Class`

### getMembers(String, String)

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

#### Signature

```
public static ConnectApi.GroupMemberPage getMembers(String communityId, String groupId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

#### Return Value

Type: `ConnectApi.GroupMemberPage Class`

### getMembers(String, String, Integer, Integer)

Returns the specified page of information about all members of the specified group.

#### API Version

28.0

#### Signature

```
public static ConnectApi.GroupMemberPage getMembers(String communityId, String groupId,
Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### getMyChatterSettings(String, String)

Returns the context user's Chatter settings for the specified group.

#### API Version

28.0

#### Signature

```
public static ConnectApi.GroupChatterSettings getMyChatterSettings(String communityId,
String groupId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

#### Return Value

Type: `ConnectApi.GroupChatterSettings Class`

## getPhoto(String, String)

Returns information about the photo for the specified group.

### API Version

28.0

### Signature

```
public static ConnectApi.Photo getPhoto(String communityId, String groupId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

### Return Value

Type: `ConnectApi.Photo Class`

## requestGroupMembership(String, String)

Requests membership in a private group for the context user.

### API Version

28.0

### Signature

```
public static ConnectApi.GroupMembershipRequest requestGroupMembership(String communityId,
String groupId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

### Return Value

Type: `ConnectApi.GroupMembershipRequest Class`

#### 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(String, String)

Returns the first page of groups that match the specified search criteria. The page contains the default number of items.

#### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterGroupPage searchGroups(String communityId, String q)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

**See Also:**

*setTestSearchGroups(String, String, ConnectApi.ChatterGroupPage)*
*Testing ConnectApi Code*

### searchGroups(String, String, Integer, Integer)

Returns the specified page of groups that match the specified search criteria.

#### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterGroupPage searchGroups(String communityId, String q, Integer
pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.ChatterGroupPage Class`

**See Also:**

*setTestSearchGroups(String, String, Integer, Integer, ConnectApi.ChatterGroupPage)*
*Testing ConnectApi Code*

### searchGroups(String, String, ConnectApi.GroupArchiveStatus, Integer, Integer)

Returns the specified page of groups that match the specified search criteria and that have the specified archive status.

#### API Version

29.0

#### Signature

```
public static ConnectApi.ChatterGroupPage searchGroups(String communityId, String q,
ConnectApi.GroupArchiveStatus archiveStatus, Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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` on page 606

*`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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.ChatterGroupPage Class`

### See Also:

*setTestSearchGroups(String, String, ConnectApi.GroupArchiveStatus, Integer, Integer, ConnectApi.ChatterGroupPage)*
*Testing ConnectApi Code*

## setPhoto(String, String, String, Integer)

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

#### Signature

```
public static ConnectApi.Photo setPhoto(String communityId, String groupId, String fileId,
Integer versionNumber)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

#### 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(String, String, ConnectApi.BinaryInput)

Sets the group photo to the specified blob..

#### API Version

28.0

#### Signature

```
public static ConnectApi.Photo setPhoto(String communityId, String groupId,
ConnectApi.BinaryInput fileUpload)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

***fileUpload***

Type: `ConnectApi.BinaryInput Class`

A file to use as the photo. The content type must be usable as an image.

### Return Value

Type: ConnectApi.Photo Class

### 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(String, String, ConnectApi.PhotoInput)

Sets and crops an already uploaded file as the group photo.

### API Version

29.0

### Signature

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String groupId,
ConnectApi.PhotoInput photo)
```

### Parameters

***communityId***

Type: String

Use null.

***groupId***

Type: String

The ID for a group.

***photo***

Type: ConnectApi.PhotoInput Class

A ConnectApi.PhotoInput object that specifies the ID and version of the file, and how to crop the file.

### Return Value

Type: ConnectApi.Photo Class

### 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(String, String, ConnectApi.PhotoInput, ConnectApi.BinaryInput)

Sets and crops a binary input as the group photo.

### API Version

29.0

### Signature

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String groupId,
ConnectApi.PhotoInput photo, ConnectApi.BinaryInput fileUpload)
```

### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

***photo***

Type: `ConnectApi.PhotoInput Class`

A `ConnectApi.PhotoInput` object that specifies how to crop the file specified in *`fileUpload`*.

***fileUpload***

Type: `ConnectApi.BinaryInput Class`

A file to use as the photo. The content type must be usable as an image.

### Return Value

Type: `ConnectApi.Photo Class`

### 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.

## setTestSearchGroups(String, String, ConnectApi.ChatterGroupPage)

Registers a `ConnectApi.ChatterGroupPage` object to be returned when the matching `ConnectApi.searchGroups` method is called in a test context. You must 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 `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 Class`

The test `ConnectApi.ChatterGroupPage` object.

### Return Value

Type: Void

## See Also:

*searchGroups(String, String)*
*Testing ConnectApi Code*

## setTestSearchGroups(String, String, Integer, Integer, ConnectApi.ChatterGroupPage)

Registers a `ConnectApi.ChatterGroupPage` object to be returned when the matching `ConnectApi.searchGroups` method is called in a test context. You must 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***result***

Type: `ConnectApi.ChatterGroupPage Class`

The test `ConnectApi.ChatterGroupPage` object.

#### Return Value

Type: Void

### See Also:

*searchGroups(String, String, Integer, Integer)*
*Testing ConnectApi Code*

## setTestSearchGroups(String, String, ConnectApi.GroupArchiveStatus, Integer, Integer, ConnectApi.ChatterGroupPage)

Registers a `ConnectApi.ChatterGroupPage` object to be returned when the matching `ConnectApi.searchGroups` method is called in a test context. You must 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 `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` on page 606

*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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***result***

Type: `ConnectApi.ChatterGroupPage Class`

The test `ConnectApi.ChatterGroupPage` object.

#### Return Value

Type: Void

### See Also:

*searchGroups(String, String, ConnectApi.GroupArchiveStatus, Integer, Integer)*
*Testing ConnectApi Code*

## updateGroup(String, String, ConnectApi.ChatterGroupInput)

Update the settings of a group.

#### API Version

28.0

#### Signature

```
public static ConnectApi.ChatterGroup updateGroup(String communityId, String groupId,
ConnectApi.ChatterGroupInput groupInput)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

***groupInput***

Type: `ConnectApi.ChatterGroupInput Class`

A `ConnectApi.ChatterGroupInput` object.

### Return Value

Type: ConnectApi.ChatterGroup Class

### 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(String, String, ConnectApi.GroupMembershipType)

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

### Signature

```
public static ConnectApi.ChatterGroup updateGroupMember(String communityId, String
membershipId, ConnectApi.GroupMembershipType role)
```

### Parameters

***communityId***

Type: String

Use 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 Class

## updateMyChatterSettings(String, String, ConnectApi.GroupEmailFrequency)

Updates the context user's Chatter settings for the specified group.

### API Version

28.0

### Signature

```
public static ConnectApi.GroupChatterSettings updateMyChatterSettings(String communityId,
String groupId, ConnectApi.GroupEmailFrequency emailFrequency)
```

### Parameters

***communityId***

Type: String

Use `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(String, String, ConnectApi.GroupEmailFrequency).

### Return Value

Type: `ConnectApi.GroupChatterSettings Class`

## updateRequestStatus(String, String, ConnectApi.GroupMembershipRequestStatus)

Updates a request to join a private group.

### API Version

28.0

### Signature

```
public static ConnectApi.GroupMembershipRequest updateRequestStatus(String communityId,
String requestId, ConnectApi.GroupMembershipRequestStatus status)
```

### Parameters

***communityId***

Type: String

Use `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 Class`

### 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);
```

# 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.

### deletePhoto(String, String)

Deletes the specified user's photo.

### follow(String, String, String)

Adds the specified *userId* as a follower to the specified *subjectId*.

### getChatterSettings(String, String)

Returns information about the default Chatter settings for the specified user.

### getFollowers(String, String)

Returns the first page of followers for the specified user ID. The page contains the default number of items.

### getFollowers(String, String, Integer, Integer)

Returns the specified page of followers for the specified user ID.

### getFollowings(String, String)

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(String, String, Integer)

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(String, String, Integer, Integer)

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(String, String, String)

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(String, String, String, Integer)

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(String, String, String, Integer, Integer)

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(String, String)

Returns the first page of groups the specified user is a member of.

### getGroups(String, String, Integer, Integer)

Returns the specified page of groups the specified user is a member of.

### getPhoto(String, String)

Returns information about the specified user's photo.

### getUser(String, String)

Returns information about the specified user.

### getUsers(String)

Returns the first page of users. The page contains the default number of items.

***getUsers(String, Integer, Integer)***

Returns the specified page of users.

***searchUserGroups(String, String, String)***

Returns the first page of groups that match the specified search criteria.

***searchUserGroups(String, String, String, Integer, Integer)***

Returns the specified page of users that matches the specified search criteria.

***searchUsers(String, String)***

Returns the first page of users that match the specified search criteria. The page contains the default number of items.

***searchUsers(String, String, Integer, Integer)***

Returns the specified page of users that match the specified search criteria.

***searchUsers(String, String, String, Integer, Integer)***

Returns the specified page of users that match the specified search criteria.

***setPhoto(String, String, String, Integer)***

Sets the user photo to be the specified, already uploaded file.

***setPhoto(String, String, ConnectApi.BinaryInput)***

Sets the provided blob to be the photo for the specified user. The content type must be usable as an image.

***setPhotoWithAttributes(String, String, ConnectApi.Photo)***

Sets and crops the existing file as the photo for the specified user. The content type must be usable as an image.

***setPhotoWithAttributes(String, String, ConnectApi.Photo, ConnectApi.BinaryInput)***

Sets and crops the provided blob as the photo for the specified user. The content type must be usable as an image.

***setTestSearchUsers(String, String, ConnectApi.UserPage)***

Registers a `ConnectApi.UserPage` object to be returned when the matching `ConnectApi.searchUsers` method is called in a test context. You must use the method with the same parameters or you receive an exception.

***setTestSearchUsers(String, String, Integer, Integer, ConnectApi.UserPage)***

Registers a `ConnectApi.UserPage` object to be returned when the matching `ConnectApi.searchUsers` method is called in a test context. You must use the method with the same parameters or you receive an exception.

***setTestSearchUsers(String, String, String, Integer, Integer, ConnectApi.UserPage)***

Registers a `ConnectApi.UserPage` object to be returned when the matching `ConnectApi.searchUsers` method is called in a test context. You must use the method with the same parameters or you receive an exception.

***updateChatterSettings(String, String, ConnectApi.GroupEmailFrequency)***

Updates the default Chatter settings for the specified user.

***updateUser(String, String, ConnectApi.UserInput)***

Updates the “About Me” section for a user.

## deletePhoto(String, String)

Deletes the specified user’s photo.

#### API Version

28.0

#### Signature

```
public static Void deletePhoto(String communityId, String userId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for the context user or the keyword `me`.

#### Return Value

Type: Void

### follow(String, String, String)

Adds the specified *`userId`* as a follower to the specified *`subjectId`*.

#### API Version

28.0

#### Signature

```
public static ConnectApi.Subscription follow(String communityId, String userId, String
subjectId)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

## getChatterSettings(String, String)

Returns information about the default Chatter settings for the specified user.

### API Version

28.0

### Signature

```
public static ConnectApi.UserChatterSettings getChatterSettings(String communityId, String userId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for the context user or the keyword `me`.

### Return Value

Type: `ConnectApi.UserChatterSettings Class`

## getFollowers(String, String)

Returns the first page of followers for the specified user ID. The page contains the default number of items.

### API Version

28.0

### Signature

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String userId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for a user.

### Return Value

Type: `ConnectApi.FollowerPage Class`

## getFollowers(String, String, Integer, Integer)

Returns the specified page of followers for the specified user ID.

### API Version

28.0

### Signature

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String userId, Integer pageParam, Integer pageSize)
```

### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

### Return Value

Type: `ConnectApi.FollowerPage Class`

## getFollowings(String, String)

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

### Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for a user.

#### Return Value

Type: `ConnectApi.FollowingPage Class`

### getFollowings(String, String, Integer)

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

#### Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
Integer pageParam)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### getFollowings(String, String, Integer, Integer)

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

#### Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.FollowingPage Class`

### getFollowings(String, String, String)

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

#### Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
String filterType)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### getFollowings(String, String, String, Integer)

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

#### Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
String filterType, Integer pageParam)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### getFollowings(String, String, String, Integer, Integer)

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

#### Signature

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
String filterType, Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.FollowingPage Class`

### getGroups(String, String)

Returns the first page of groups the specified user is a member of.

#### API Version

28.0

#### Signature

```
public static ConnectApi.UserGroupPage getGroups(String communityId, String userId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for a user.

#### Return Value

Type: `ConnectApi.UserGroupPage Class`

## getGroups(String, String, Integer, Integer)

Returns the specified page of groups the specified user is a member of.

### API Version

28.0

### Signature

```
public static ConnectApi.UserGroupPage getGroups(String communityId, String userId, Integer pageParam, Integer pageSize)
```

### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

### Return Value

Type: `ConnectApi.UserGroupPage Class`

## getPhoto(String, String)

Returns information about the specified user's photo.

### API Version

28.0

### Signature

```
public static ConnectApi.Photo getPhoto(String communityId, String userId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for a user.

#### Return Value

Type: `ConnectApi.Photo Class`

### getUser(String, String)

Returns information about the specified user.

#### API Version

28.0

#### Signature

```
public static ConnectApi.UserSummary getUser(String communityId, String userId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for a user.

#### Return Value

Type: `ConnectApi.UserSummary Class`

#### Usage

If the context user is not a Chattercustomer, the returned object can be downcast to a UserDetail object.

### getUsers(String)

Returns the first page of users. The page contains the default number of items.

#### API Version

28.0

#### Signature

```
public static ConnectApi.UserPage getUsers(String communityId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

#### Return Value

Type: ConnectApi.UserPage Class

### getUsers(String, Integer, Integer)

Returns the specified page of users.

#### API Version

28.0

#### Signature

```
public static ConnectApi.UserPage getUsers(String communityId, Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use 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 items per page. Valid values are between 1 and 100. If you pass in null, the default size is 25.

#### Return Value

Type: ConnectApi.UserPage Class

### searchUserGroups(String, String, String)

Returns the first page of groups that match the specified search criteria.

#### API Version

30.0

#### Signature

```
public static ConnectApi.UserGroupPage searchUserGroups(String communityId, String userId, String q)
```

#### Parameters

***communityId***

Type: String

Use null.

***userId***

Type: String

The ID for the context user or the keyword `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.UserGroupPage Class`

A paginated list of groups the context user is a member of.

### searchUserGroups(String, String, String, Integer, Integer)

Returns the specified page of users that matches the specified search criteria.

#### API Version

30.0

#### Signature

```
public static ConnectApi.UserGroupPage searchUserGroups(String communityId, String userId,
String q, Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for the context user or the keyword `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.

***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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.UserGroupPage Class`

A paginated list of groups the context user is a member of.

### searchUsers(String, String)

Returns the first page of users that match the specified search criteria. The page contains the default number of items.

#### API Version

28.0

#### Signature

```
public static ConnectApi.UserPage searchUsers(String communityId, String q)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### searchUsers(String, String, Integer, Integer)

Returns the specified page of users that match the specified search criteria.

#### API Version

28.0

#### Signature

```
public static ConnectApi.UserPage searchUsers(String communityId, String q, Integer pageParam,
Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.UserPage Class`

### searchUsers(String, String, String, Integer, Integer)

Returns the specified page of users that match the specified search criteria.

#### API Version

28.0

#### Signature

```
public static ConnectApi.UserPage searchUsers(String communityId, String q, String
searchContextId, Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: ConnectApi.UserPage Class

### setPhoto(String, String, String, Integer)

Sets the user photo to be the specified, already uploaded file.

#### API Version

28.0

#### Signature

```
public static ConnectApi.Photo setPhoto(String communityId, String userId, String fileId,
Integer versionNumber)
```

#### Parameters

***communityId***

Type: String

Use 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 Class

#### Usage

Photos are processed asynchronously and may not be visible right away.

### setPhoto(String, String, ConnectApi.BinaryInput)

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

#### Signature

```
public static ConnectApi.Photo setPhoto(String communityId, String userId,
ConnectApi.BinaryInput fileUpload)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for the context user or the keyword `me`.

***fileUpload***

Type: `ConnectApi.BinaryInput Class`

A file to use as the photo. The content type must be usable as an image.

#### Return Value

Type: `ConnectApi.Photo Class`

#### Usage

Photos are processed asynchronously and may not be visible right away.

### setPhotoWithAttributes(String, String, ConnectApi.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

#### Signature

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String userId,
ConnectApi.Photo photo)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for the context user or the keyword `me`.

***photo***

Type: `ConnectApi.Photo Class`

A `ConnectApi.PhotoInput` object specifying the file ID, version number, and cropping parameters.

#### Return Value

Type: `ConnectApi.Photo Class`

### Usage

Photos are processed asynchronously and may not be visible right away.

## setPhotoWithAttributes(String, String, ConnectApi.Photo, ConnectApi.BinaryInput)

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

### Signature

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String userId,
ConnectApi.Photo photo, ConnectApi.BinaryInput fileUpload)
```

### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for the context user or the keyword `me`.

***photo***

Type: `ConnectApi.Photo Class`

A `ConnectApi.PhotoInput` object specifying the cropping parameters.

***fileUpload***

Type: `ConnectApi.BinaryInput Class`

A file to use as the photo. The content type must be usable as an image.

### Return Value

Type: `ConnectApi.Photo Class`

### Usage

Photos are processed asynchronously and may not be visible right away.

## setTestSearchUsers(String, String, ConnectApi.UserPage)

Registers a `ConnectApi.UserPage` object to be returned when the matching `ConnectApi.searchUsers` method is called in a test context. You must 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 `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 Class`

The object containing test data.

### Return Value

Type: Void

## setTestSearchUsers(String, String, Integer, Integer, ConnectApi.UserPage)

Registers a `ConnectApi.UserPage` object to be returned when the matching `ConnectApi.searchUsers` method is called in a test context. You must 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***result***

Type: `ConnectApi.UserPage Class`

The object containing test data.

#### Return Value

Type: Void

### setTestSearchUsers(String, String, String, Integer, Integer, ConnectApi.UserPage)

Registers a `ConnectApi.UserPage` object to be returned when the matching `ConnectApi.searchUsers` method is called in a test context. You must 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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***result***

Type: `ConnectApi.UserPage Class`

The object containing test data.

#### Return Value

Type: Void

### updateChatterSettings(String, String, ConnectApi.GroupEmailFrequency)

Updates the default Chatter settings for the specified user.

#### API Version

28.0

#### Signature

```
public static ConnectApi.UserChatterSettings updateChatterSettings(String communityId,
String userId, ConnectApi.GroupEmailFrequency defaultGroupEmailFrequency)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### updateUser(String, String, ConnectApi.UserInput)

Updates the "About Me" section for a user.

#### API Version

29.0

#### Signature

```
public static ConnectApi.UserDetail updateUser(String communityId, String userId,
ConnectApi.UserInput userInput)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for the context user or the keyword `me`.

***userInput***

Type: `ConnectApi.UserInput Class`

Specifies the updated information.

#### Return Value

Type: `ConnectApi.UserDetail Class`

## Communities Class

Access general information about communities in your organization.

### Namespace

ConnectApi

## Communities Methods

The following are methods for `Communities`. All methods are static.

***getCommunities()***

Returns a list of communities the context user has access to.

***getCommunities(ConnectApi.CommunityStatus)***

Returns a list of communities the context user has access to with the specified status.

***getCommunity(String)***

Returns information about the specific community.

### getCommunities()

Returns a list of communities the context user has access to.

#### API Version

28.0

#### Signature

```
public static ConnectApi.CommunityPage getCommunities()
```

#### Return Value

Type: ConnectApi.CommunityPage Class

### getCommunities(ConnectApi.CommunityStatus)

Returns a list of communities the context user has access to with the specified status.

#### API Version

28.0

#### Signature

```
public static ConnectApi.CommunityPage getCommunities(ConnectApi.CommunityStatus
communityStatus)
```

#### Parameters

***communityStatus***

Type: ConnectApi.CommunityStatus

*communityStatus*—Specifies the current status of the community. Values are:

- Live
- Inactive
- UnderConstruction

#### Return Value

Type: ConnectApi.CommunityPage Class

### getCommunity(String)

Returns information about the specific community.

#### API Version

28.0

#### 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 Class

# 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.

#### *addFlagToComment(String, String)*

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(String, String, ConnectApi.CommunityFlagVisibility)*

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.

#### *addFlagToFeedItem(String, String)*

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(String, String, ConnectApi.CommunityFlagVisibility)*

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(String, String)*

Get the moderation flags on a comment. To get the flags, the logged-in user must have the “Moderate Communities Feeds” permission.

#### *getFlagsOnComment(String, String, ConnectApi.CommunityFlagVisibility)*

Get the moderation flags with specified visibility on a comment. To get the flags, the logged-in user must have the “Moderate Communities Feeds” permission.

#### *getFlagsOnFeedItem(String, String)*

Get the moderation flags on a feed item. To get the flags, the logged-in user must have the “Moderate Communities Feeds” permission.

#### *getFlagsOnFeedItem(String, String, ConnectApi.CommunityFlagVisibility)*

Get the moderation flags with specified visibility on a feed item. To get the flags, the logged-in user must have the “Moderate Communities Feeds” permission.

#### *removeFlagsOnComment(String, String, String)*

Remove the moderation flags from a comment. To remove a flag from a comment the logged-in user must have added the flag or must have the “Moderate Communities Feeds” permission.

### *removeFlagsOnFeedItem(String, String, String)*

Remove the moderation flags from a feed item. To remove a flag from a feed item, the logged-in user must have added the flag or must have the “Moderate Communities Feeds” permission.

## addFlagToComment(String, String)

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

#### Signature

```
public static ConnectApi.ModerationFlags addFlagToComment(String communityId, String commentId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***commentId***

Type: String

The ID for a comment.

#### Return Value

Type: ConnectApi.ModerationFlags Class

## addFlagToComment(String, String, ConnectApi.CommunityFlagVisibility)

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

#### Signature

```
public static ConnectApi.ModerationFlags addFlagToComment(String communityId, String commentId, ConnectApi.CommunityFlagVisibility visibility)
```

#### Parameters

***communityId***

Type: String

Use `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 item.
- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged item.

#### Return Value

Type: ConnectApi.ModerationFlags Class

### addFlagToFeedItem(String, String)

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

#### Signature

```
public static ConnectApi.ModerationFlags addFlagToFeedItem(String communityId, String feedItemId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

#### Return Value

Type: ConnectApi.ModerationFlags Class

### addFlagToFeedItem(String, String, ConnectApi.CommunityFlagVisibility)

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

### Signature

```
public static ConnectApi.ModerationFlags addFlagToFeedItem(String communityId, String feedItemId, ConnectApi.CommunityFlagVisibility visibility)
```

### Parameters

***communityId***

Type: String

Use `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 item.
- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged item.

### Return Value

Type: ConnectApi.ModerationFlags Class

## getFlagsOnComment(String, String)

Get the moderation flags on a comment. To get the flags, the logged-in user must have the “Moderate Communities Feeds” permission.

### API Version

29.0

### Signature

```
public static ConnectApi.ModerationFlags getFlagsOnComment(String communityId, String commentId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***commentId***

Type: String

The ID for a comment.

### Return Value

Type: ConnectApi.ModerationFlags Class

## getFlagsOnComment(String, String, ConnectApi.CommunityFlagVisibility)

Get the moderation flags with specified visibility on a comment. To get the flags, the logged-in user must have the “Moderate Communities Feeds” permission.

### API Version

30.0

### Signature

```
public static ConnectApi.ModerationFlags getFlagsOnComment(String communityId, String commentId, ConnectApi.CommunityFlagVisibility visibility)
```

### Parameters

***communityId***

Type: String

Use `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 item.
- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged item.

### Return Value

Type: ConnectApi.ModerationFlags Class

## getFlagsOnFeedItem(String, String)

Get the moderation flags on a feed item. To get the flags, the logged-in user must have the “Moderate Communities Feeds” permission.

### API Version

29.0

### Signature

```
public static ConnectApi.ModerationFlags getFlagsOnFeedItem(String communityId, String feedItemId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

#### Return Value

Type: ConnectApi.ModerationFlags Class

### getFlagsOnFeedItem(String, String, ConnectApi.CommunityFlagVisibility)

Get the moderation flags with specified visibility on a feed item. To get the flags, the logged-in user must have the "Moderate Communities Feeds" permission.

#### API Version

30.0

#### Signature

```
public static ConnectApi.ModerationFlags getFlagsOnFeedItem(String communityId, String feedItemId, ConnectApi.CommunityFlagVisibility visibility)
```

#### Parameters

***communityId***

Type: String

Use `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 item.
- `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged item.

#### Return Value

Type: ConnectApi.ModerationFlags Class

### removeFlagsOnComment(String, String, String)

Remove the moderation flags from a comment. To remove a flag from a comment the logged-in user must have added the flag or must have the "Moderate Communities Feeds" permission.

#### API Version

29.0

### Signature

```
public static ConnectApi.ModerationFlags removeFlagsOnComment(String communityId, String commentId, String userId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***commentId***

Type: String

The ID for a comment.

***userId***

Type: String

The ID for a user.

### Return Value

Type: Void

## removeFlagsOnFeedItem(String, String, String)

Remove the moderation flags from a feed item. To remove a flag from a feed item, the logged-in user must have added the flag or must have the “Moderate Communities Feeds” permission.

### API Version

29.0

### Signature

```
public static ConnectApi.ModerationFlags removeFlagsOnFeedItem(String communityId, String feedItemId, String userId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***feedItemId***

Type: String

The ID for a feed item.

***userId***

Type: String

The ID for a user.

#### Return Value

Type: Void

# 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.

### *getSettings()*

Returns information about the organization and logged-in user, including which features are enabled.

### getSettings()

Returns information about the organization and logged-in user, including which features are enabled.

#### API Version

28.0

#### Signature

```
public static ConnectApi. OrganizationSettings getSettings()
```

#### Return Value

Type: `ConnectApi. OrganizationSettings`

# 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.

### *getMentionCompletions(String, String, String)*

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.

***getMentionCompletions(String, String, String, ConnectApi.MentionCompletionType, Integer, Integer)***

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.

***getMentionValidations(String, String, List<String>, ConnectApi.FeedItemVisibilityType)***

Information about whether the specified mentions are valid for the context user.

***setTestGetMentionCompletions(String, String, String, ConnectApi.MentionCompletionPage)***

Registers a `ConnectApi.MentionCompletionPage` object to be returned when `getMentionCompletions(String, String, String)` is called in a test context.

***setTestGetMentionCompletions(String, String, String, ConnectApi.MentionCompletionType, Integer, Integer, ConnectApi.MentionCompletionPage)***

Registers a `ConnectApi.MentionCompletionPage` object to be returned when `getMentionCompletions(String, String, String, ConnectApi.MentionCompletionType, Integer, Integer)` is called in a test context.

## getMentionCompletions(String, String, String)

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

### Signature

```
public static ConnectApi.MentionCompletionPage getMentionCompletions (String communityId, String q, String contextId)
```

### Parameters

***communityId***

Type: String

Use `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 Class`

### 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(String, String, String, ConnectApi.MentionCompletionPage)*
*Testing ConnectApi Code*

## getMentionCompletions(String, String, String, ConnectApi.MentionCompletionType, Integer, Integer)

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

### Signature

```
public static ConnectApi.Mentions getMentionCompletions (String communityId, String q,
String contextId, ConnectApi.MentionCompletionType type, Integer pageParam, Integer pageSize)
```

### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.MentionCompletionPage Class`

#### 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(String, String, String, ConnectApi.MentionCompletionType, Integer, Integer, ConnectApi.MentionCompletionPage)*
*Testing ConnectApi Code*

### getMentionValidations(String, String, List<String>, ConnectApi.FeedItemVisibilityType)

Information about whether the specified mentions are valid for the context user.

#### API Version

29.0

#### Signature

```
public static ConnectApi.Mentions getMentionValidations(String communityId, String parentId,
List<String> recordIds, ConnectApi.FeedItemVisibilityType visibility)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### 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`.

## setTestGetMentionCompletions(String, String, String, ConnectApi.MentionCompletionPage)

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 `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 Class

A `ConnectApi.MentionCompletionPage` object containing test data.

### Return Value

Type: Void

### See Also:

*getMentionCompletions(String, String, String)*
*Testing ConnectApi Code*

## setTestGetMentionCompletions(String, String, String, ConnectApi.MentionCompletionType, Integer, Integer, ConnectApi.MentionCompletionPage)

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 `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

***result***

Type: ConnectApi.MentionCompletionPage Class

A `ConnectApi.MentionCompletionPage` object containing test data.

#### Return Value

Type: Void

**See Also:**

*getMentionCompletions(String, String, String, ConnectApi.MentionCompletionType, Integer, Integer)*
*Testing ConnectApi Code*

# RecordDetails Class

Access information about records in your organization.

### Namespace

ConnectApi

## RecordDetails Methods

The following are methods for `RecordDetails`. All methods are static.

***getRecentRecords(String, String)***

Returns a list of objects that contain summary information about records recently added, edited, or viewed by the context user.

***getRecentRecords(String, String, Integer)***

Returns an object that contains a list populated by the specified number of objects. Each object contains summary information about a record recently added, edited, or viewed by the context user.

***getRecordView(String, String)***

Returns a `RecordView` object, which contains a combined view of layout information (metadata) and data for the specified record.

### getRecentRecords(String, String)

Returns a list of objects that contain summary information about records recently added, edited, or viewed by the context user.

#### API Version

30.0

#### Signature

```
public static ConnectApi.RecordSummaryList getRecentRecords(String communityId, String
userId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for the context user or the keyword `me`.

#### Return Value

Type: ConnectApi.RecordSummaryList

An object that contains a list of record objects. The records can be of any type in the organization, including custom objects. The record objects are summary objects, not detail objects.

### getRecentRecords(String, String, Integer)

Returns an object that contains a list populated by the specified number of objects. Each object contains summary information about a record recently added, edited, or viewed by the context user.

#### API Version

30.0

#### Signature

```
public static ConnectApi.RecordSummaryList getRecentRecords(String communityId, String
userId, Integer size)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for the context user or the keyword `me`.

***size***

Type: Integer

The maximum number of recently used objects to return between 1–150. The default value is 10.

#### Return Value

Type: ConnectApi.RecordSummaryList

An object that contains a list of record objects. The records can be of any type in the organization, including custom objects. The record objects are summary objects, not detail objects.

### getRecordView(String, String)

Returns a `RecordView` object, which contains a combined view of layout information (metadata) and data for the specified record.

| Available in: **Unlimited**, **Enterprise**, and **Developer** Editions |
|---|

### API Version

30.0

### Signature

```
public static ConnectApi.RecordView getRecordView(String communityId, String recordId)
```

### Parameters

***communityId***

Type: String

Use `null`.

***recordId***

Type: String

The ID of a record.

### Return Value

Type: 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.

### Usage

This method uses a whitelist to determine whether to return information about a record. If the object type in the *recordId* argument is not in the whitelist, the call returns the error `"ConnectApi.ConnectApiException: Object type not supported"`. These are the supported objects:

| Object Type | Available |
|---|---|
| Custom objects | 30.0 |
| Account | 30.0 |
| Campaign | 30.0 |
| Case | 30.0 |
| CollaborationGroup | 30.0 |
| Contact | 30.0 |
| ContentDocument | 30.0 |
| ContentVersion | 30.0 |
| Contract | 30.0 |
| Dashboard | 30.0 |
| Event | 30.0 |
| Lead | 30.0 |
| LiveChatTranscript | 30.0 |
| Opportunity | 30.0 |
| OpportunityLineItem | 30.0 |

| Object Type | Available |
|---|---|
| Product | 30.0 |
| ServiceContract | 30.0 |
| Solution | 30.0 |
| Task | 30.0 |
| User | 30.0 |

# Records Class

Access information about record motifs, which are small icons used to distinguish record types in the salesforce.com UI.

### Namespace

ConnectApi

## Records Methods

The following are methods for `Records`. All methods are static.

### *getMotif(String, String)*

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.

### getMotif(String, String)

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

#### Signature

```
public static ConnectApi.Motif getMotif(String communityId, String idOrPrefix)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***idOrPrefix***

Type: String

An ID or key prefix.

#### Return Value

Type: `ConnectApi.Motif`

#### Usage

Each Database.com record type has its own set of motif icons. See `ConnectApi.Motif`.

# 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, and add and remove topics from records and feed items.

## Namespace

ConnectApi

## Topics Methods

The following are methods for `Topics`. All methods are static.

### *assignTopic(String, String, String)*

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(String, String, String)*

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(String, String)*

Deletes the specified topic. Only users with the "Delete Topics" or "Modify All Data" permission can delete topics.

### *getGroupsRecentlyTalkingAboutTopic(String, String)*

Returns information about the five groups that most recently contributed to the specified topic.

### *getRecentlyTalkingAboutTopicsForGroup(String, String)*

Returns up to five topics most recently used in the specified group.

### *getRecentlyTalkingAboutTopicsForUser(String, String)*

Topics recently used by the specified user. Get up to five topics most recently used by the specified user.

### *getRelatedTopics(String, String)*

List of five topics most closely related to the specified topic.

### *getTopic(String, String)*

Returns information about the specified topic.

### *getTopics(String, String)*

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(String)*

Returns the first page of topics for the organization.

### *getTopics(String, ConnectApi.TopicSort)*

Returns the first page of topics for the organization in the specified order.

### *getTopics(String, Integer, Integer)*

Returns the topics for the specified page.

### *getTopics(String, Integer, Integer, ConnectApi.TopicSort)*

Returns the topics for the specified page in the specified order.

### *getTopics(String, String, ConnectApi.TopicSort)*

Returns the topics that match the specified search criteria in the specified order.

### *getTopics(String, String, Integer, Integer)*

Returns the topics that match the specified search criteria for the specified page.

### *getTopics(String, String, Integer, Integer, ConnectApi.TopicSort)*

Returns the topics that match the specified search criteria for the specified page in the specified order.

### *getTopicSuggestions(String, String, Integer)*

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(String, String)*

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(String, String, Integer)*

Returns suggested topics for the specified string of text.

### *getTopicSuggestionsForText(String, String)*

Returns suggested topics for the specified string of text.

### *getTrendingTopics(String)*

List of the top five trending topics for the organization.

### *getTrendingTopics(String, Integer)*

List of the top five trending topics for the organization.

### *setTestGetGroupsRecentlyTalkingAboutTopic(String, String, ConnectApi.ChatterGroupSummaryPage)*

Registers a `ConnectApi.ChatterGroupSummaryPage` object to be returned when `ConnectApi.getGroupsRecentlyTalkingAboutTopic` is called in a test context.

### *setTestGetRecentlyTalkingAboutTopicsForGroup(String, String, ConnectApi.TopicPage)*

Registers a `ConnectApi.TopicPage` object to be returned when the `ConnectApi.getRecentlyTalkingAboutTopicsForGroup` method is called in a test context.

### *setTestGetRecentlyTalkingAboutTopicsForUser(String, String, ConnectApi.TopicPage)*

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. You must use the method with the same parameters or you receive an exception.

#### *setTestGetRelatedTopics(String, String, ConnectApi.TopicPage)*

Registers a `ConnectApi.TopicPage` object to be returned when the `ConnectApi.getRelatedTopics` method is called in a test context.

#### *setTestGetTopicSuggestions(String, String, Integer, ConnectApi.TopicSuggestionPage)*

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestions` method is called in a test context. You must use the method with the same parameters or you receive an exception. You must use the method with the same parameters or you receive an exception.

#### *setTestGetTopicSuggestions(String, String, ConnectApi.TopicSuggestionPage)*

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestions` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### *setTestGetTopicSuggestionsForText(String, String, Integer, ConnectApi.TopicSuggestionPage)*

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestionsForText` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### *setTestGetTopicSuggestionsForText(String, String, ConnectApi.TopicSuggestionPage)*

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestionsForText` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### *setTestGetTrendingTopics(String, ConnectApi.TopicPage)*

Registers a `ConnectApi.TopicPage` object to be returned when the matching `ConnectApi.getTrendingTopics` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### *setTestGetTrendingTopics(String, Integer, ConnectApi.TopicPage)*

Registers a `ConnectApi.TopicPage` object to be returned when the matching `ConnectApi.getTrendingTopics` method is called in a test context. You must use the method with the same parameters or you receive an exception.

#### *unassignTopic(String, String, String)*

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(String, String, ConnectApi.TopicInput)*

Updates the description or spacing and capitalization of the name of the specified topic. Only users with the “Edit Topics” permission can edit topic names and descriptions.

## assignTopic(String, String, String)

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

### Signature

```
public static ConnectApi.Topic assignTopic(String communityId, String recordId, String topicId)
```

### Parameters

***communityId***

Type: String

Use `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 Class`

## assignTopicByName(String, String, String)

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

### Signature

```
public static ConnectApi.Topic assignTopicByName(String communityId, String recordId, String topicName)
```

### Parameters

***communityId***

Type: String

Use `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 Class

### deleteTopic(String, String)

Deletes the specified topic. Only users with the “Delete Topics” or “Modify All Data” permission can delete topics.

#### API Version

29.0

#### Signature

```
public static Void deleteTopic(String, communityId, String topicId)
```

#### Parameters

***communityId***

Type: String,

Use null.

***topicId***

Type: String

The ID for a topic.

#### Return Value

Type: Void

### getGroupsRecentlyTalkingAboutTopic(String, String)

Returns information about the five groups that most recently contributed to the specified topic.

#### API Version

29.0

#### Signature

```
public static ConnectApi.ChatterGroupSummaryPage getGroupsRecentlyTalkingAboutTopic(String
communityId, String topicId)
```

#### Parameters

***communityId***

Type: String

Use null.

***topicId***

Type: String

The ID for a topic.

#### Return Value

Type: `ConnectApi.ChatterGroupSummaryPage Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

**See Also:**

*setTestGetGroupsRecentlyTalkingAboutTopic(String, String, ConnectApi.ChatterGroupSummaryPage)*
*Testing ConnectApi Code*

### getRecentlyTalkingAboutTopicsForGroup(String, String)

Returns up to five topics most recently used in the specified group.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getRecentlyTalkingAboutTopicsForGroup(String communityId,
String groupId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***groupId***

Type: String

The ID for a group.

#### Return Value

Type: `ConnectApi.TopicPage Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

**See Also:**

*setTestGetRecentlyTalkingAboutTopicsForGroup(String, String, ConnectApi.TopicPage)*
*Testing ConnectApi Code*

### getRecentlyTalkingAboutTopicsForUser(String, String)

Topics recently used by the specified user. Get up to five topics most recently used by the specified user.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getRecentlyTalkingAboutTopicsForUser(String communityId,
String userId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***userId***

Type: String

The ID for a user.

#### Return Value

Type: `ConnectApi.TopicPage Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

**See Also:**

*setTestGetRecentlyTalkingAboutTopicsForUser(String, String, ConnectApi.TopicPage)*
*Testing ConnectApi Code*

### getRelatedTopics(String, String)

List of five topics most closely related to the specified topic.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getRelatedTopics(String communityId, String topicId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***topicId***

Type: String

The ID for a topic.

#### Return Value

Type: `ConnectApi.TopicPage Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

**See Also:**

*setTestGetRelatedTopics(String, String, ConnectApi.TopicPage)*
*Testing ConnectApi Code*

### getTopic(String, String)

Returns information about the specified topic.

#### API Version

29.0

#### Signature

```
public static ConnectApi.Topic getTopic(String communityId, String topicId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***topicId***

Type: String

The ID for a topic.

#### Return Value

Type: `ConnectApi.Topic Class`

### getTopics(String, String)

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

#### Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, String recordId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***recordId***

Type: String

The ID for a record or feed item.

#### Return Value

Type: `ConnectApi.TopicPage Class`

### getTopics(String)

Returns the first page of topics for the organization.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getTopics(String communityId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

#### Return Value

Type: `ConnectApi.TopicPage Class`

### getTopics(String, ConnectApi.TopicSort)

Returns the first page of topics for the organization in the specified order.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, ConnectApi.TopicSort
sortParam)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### getTopics(String, Integer, Integer)

Returns the topics for the specified page.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.TopicPage Class`

### getTopics(String, Integer, Integer, ConnectApi.TopicSort)

Returns the topics for the specified page in the specified order.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, Integer pageParam, Integer pageSize, ConnectApi.TopicSort sortParam)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 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 Class`

### getTopics(String, String, ConnectApi.TopicSort)

Returns the topics that match the specified search criteria in the specified order.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, String q,
ConnectApi.TopicSort sortParam)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

### getTopics(String, String, Integer, Integer)

Returns the topics that match the specified search criteria for the specified page.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, String q, Integer pageParam,
Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.TopicPage Class`

### getTopics(String, String, Integer, Integer, ConnectApi.TopicSort)

Returns the topics that match the specified search criteria for the specified page in the specified order.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getTopics(String communityId, String q, Integer pageParam,
Integer pageSize, ConnectApi.TopicSort sortParam)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 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 Class`

### getTopicSuggestions(String, String, Integer)

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

#### Signature

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestions(String communityId, String recordId, Integer maxResults)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestGetTopicSuggestions(String, String, Integer, ConnectApi.TopicSuggestionPage)*
*Testing ConnectApi Code*

## getTopicSuggestions(String, String)

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

#### Signature

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestions(String communityId, String recordId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***recordId***

Type: String

The ID for a record or feed item.

#### Return Value

Type: `ConnectApi.TopicSuggestionPage Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestGetTopicSuggestions(String, String, ConnectApi.TopicSuggestionPage)*
*Testing ConnectApi Code*

## getTopicSuggestionsForText(String, String, Integer)

Returns suggested topics for the specified string of text.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestionsForText(String communityId,
String text, Integer maxResults)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestGetTopicSuggestionsForText(String, String, Integer, ConnectApi.TopicSuggestionPage)*
*Testing ConnectApi Code*

## getTopicSuggestionsForText(String, String)

Returns suggested topics for the specified string of text.

### API Version

29.0

### Signature

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestionsForText(String communityId,
String text)
```

### Parameters

***communityId***

Type: String

Use `null`.

***text***

Type: String

String of text.

### Return Value

Type: `ConnectApi.TopicSuggestionPage Class`

### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestGetTopicSuggestionsForText(String, String, ConnectApi.TopicSuggestionPage)*
*Testing ConnectApi Code*

## getTrendingTopics(String)

List of the top five trending topics for the organization.

### API Version

29.0

### Signature

```
public static ConnectApi.TopicPage getTrendingTopics(String communityId)
```

### Parameters

***communityId***

Type: String

Use `null`.

#### Return Value

Type: `ConnectApi.TopicPage Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestGetTrendingTopics(String, ConnectApi.TopicPage)*
*Testing ConnectApi Code*

## getTrendingTopics(String, Integer)

List of the top five trending topics for the organization.

#### API Version

29.0

#### Signature

```
public static ConnectApi.TopicPage getTrendingTopics(String communityId, Integer maxResults)
```

#### Parameters

***communityId***

Type: String

Use `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 Class`

#### Usage

To test code that uses this method, use the matching set test method (prefix the method name with `setTest`). You must use the set test method with the same parameters or the code throws an exception.

### See Also:

*setTestGetTrendingTopics(String, Integer, ConnectApi.TopicPage)*
*Testing ConnectApi Code*

## setTestGetGroupsRecentlyTalkingAboutTopic(String, String, ConnectApi.ChatterGroupSummaryPage)

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 `null`.

***topicId***

Type: String

The ID for a topic.

***result***

Type: `ConnectApi.ChatterGroupSummaryPage Class`

The object containing test data.

### Return Value

Type: Void

### See Also:

*getGroupsRecentlyTalkingAboutTopic(String, String)*
*Testing ConnectApi Code*

## setTestGetRecentlyTalkingAboutTopicsForGroup(String, String, ConnectApi.TopicPage)

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 `null`.

***groupId***

Type: String

The ID for a group.

***result***

Type: `ConnectApi.TopicPage Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getRecentlyTalkingAboutTopicsForGroup(String, String)*
*Testing ConnectApi Code*

## setTestGetRecentlyTalkingAboutTopicsForUser(String, String, ConnectApi.TopicPage)

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. You must 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 `null`.

***userId***

Type: String

The ID for a user.

***result***

Type: `ConnectApi.TopicPage Class`

Specify the test topics page.

#### Return Value

Type: Void

### See Also:

*getRecentlyTalkingAboutTopicsForUser(String, String)*
*Testing ConnectApi Code*

## setTestGetRelatedTopics(String, String, ConnectApi.TopicPage)

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 `null`.

***topicId***

Type: String

The ID for a topic.

***result***

Type: `ConnectApi.TopicPage Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getRelatedTopics(String, String)*
*Testing ConnectApi Code*

## setTestGetTopicSuggestions(String, String, Integer, ConnectApi.TopicSuggestionPage)

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestions` method is called in a test context. You must use the method with the same parameters or you receive an exception. You must 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 `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 Class`

Specify the test topic suggestions page.

#### Return Value

Type: Void

### See Also:

*getTopicSuggestions(String, String, Integer)*
*Testing ConnectApi Code*

## setTestGetTopicSuggestions(String, String, ConnectApi.TopicSuggestionPage)

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestions` method is called in a test context. You must 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 `null`.

***recordId***

Type: String

The ID for a record or feed item.

***result***

Type: `ConnectApi.TopicSuggestionPage Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getTopicSuggestions(String, String)*
*Testing ConnectApi Code*

## setTestGetTopicSuggestionsForText(String, String, Integer, ConnectApi.TopicSuggestionPage)

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestionsForText` method is called in a test context. You must 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 `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 Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getTopicSuggestionsForText(String, String, Integer)*
*Testing ConnectApi Code*

## setTestGetTopicSuggestionsForText(String, String, ConnectApi.TopicSuggestionPage)

Registers a `ConnectApi.TopicSuggestionPage` object to be returned when the matching `ConnectApi.getTopicSuggestionsForText` method is called in a test context. You must 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 `null`.

***text***

Type: String

String of text.

***result***

Type: `ConnectApi.TopicSuggestionPage Class`

The object containing test data.

#### Return Value

Type: Void

### See Also:

*getTopicSuggestionsForText(String, String)*
*Testing ConnectApi Code*

### setTestGetTrendingTopics(String, ConnectApi.TopicPage)

Registers a `ConnectApi.TopicPage` object to be returned when the matching `ConnectApi.getTrendingTopics` method is called in a test context. You must 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 `null`.

***result***

Type: `ConnectApi.TopicPage Class`

The object containing test data.

#### Return Value

Type: Void

**See Also:**

*getTrendingTopics(String)*
*Testing ConnectApi Code*

### setTestGetTrendingTopics(String, Integer, ConnectApi.TopicPage)

Registers a `ConnectApi.TopicPage` object to be returned when the matching `ConnectApi.getTrendingTopics` method is called in a test context. You must 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 `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 Class`

The object containing test data.

#### Return Value

Type: Void

## See Also:

*getTrendingTopics(String, Integer)*
*Testing ConnectApi Code*

## unassignTopic(String, String, String)

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

#### Signature

```
public static Void unassignTopic(String communityId, String recordId, String topicId)
```

#### Parameters

***communityId***

Type: String

Use `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(String, String, ConnectApi.TopicInput)

Updates the description or spacing and capitalization of the name of the specified topic. Only users with the “Edit Topics” permission can edit topic names and descriptions.

#### API Version

29.0

#### Signature

```
public static ConnectApi.Topic updateTopic(String communityId, String topicId,
ConnectApi.TopicInput topic)
```

#### Parameters

***communityId***

Type: String

Use `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.

#### Return Value

Type: `ConnectApi.Topic Class`

## UserProfiles Class

Access user profile data. 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.

***getUserProfile(String, String)***

Returns the user profile of the context user.

### getUserProfile(String, String)

Returns the user profile of the context user.

#### API Version

29.0

#### Signature

```
public static ConnectApi.UserProfile getUserProfile(String communityId, String userId)
```

#### Parameters

***communityId***

Type: String

Use `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.

***getZone(String, String)***

Returns a specific zone based on the zone ID.

***getZones(String)***

Returns a paginated list of zones.

***getZones(String, Integer, Integer)***

Returns a paginated list of zones with the specified page and page size.

***searchInZone(String, String, String, ConnectApi.ZoneSearchResultType)***

Search a zone by keyword. Specify whether to search articles or questions.

***searchInZone(String, String, String, ConnectApi.ZoneSearchResultType, String, Integer)***

Search a zone by keyword. Specify whether to search articles or questions and specify the page of information to view and the page size.

#### *setTestSearchInZone(String, String, String, ConnectApi.ZoneSearchResultType, ConnectApi.ZoneSearchPage)*

Registers a `ConnectApi.ZoneSearchPage` object to be returned when `searchInZone(String, String, String, ConnectApi.ZoneSearchResultType)` is called in a test context.

#### *setTestSearchInZone(String, String, String, ConnectApi.ZoneSearchResultType, String, Integer, ConnectApi.ZoneSearchPage)*

Registers a `ConnectApi.ZoneSearchPage` object to be returned when `searchInZone(String, String, String, ConnectApi.ZoneSearchResultType, String, Integer)` is called in a test context.

### getZone(String, String)

Returns a specific zone based on the zone ID.

#### API Version

29.0

#### Signature

```
public static ConnectApi.Zone getZone(String communityId, String zoneId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

***zoneId***

Type: String

The ID of a zone.

#### Return Value

Type: `ConnectApi.Zone`

### getZones(String)

Returns a paginated list of zones.

#### API Version

29.0

#### Signature

```
public static ConnectApi.ZonePage getZones(String communityId)
```

#### Parameters

***communityId***

Type: String

Use `null`.

#### Return Value

Type: ConnectApi.ZonePage

### getZones(String, Integer, Integer)

Returns a paginated list of zones with the specified page and page size.

#### API Version

29.0

#### Signature

```
public static ConnectApi.Zone getZones(String communityId, Integer pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use 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 items per page. Valid values are between 1 and 100. If you pass in null, the default size is 25.

#### Return Value

Type: ConnectApi.Zone

### searchInZone(String, String, String, ConnectApi.ZoneSearchResultType)

Search a zone by keyword. Specify whether to search articles or questions.

#### API Version

29.0

#### Signature

```
public static ConnectApi.ZoneSearchPage searchInZone(String communityId, String zoneId, String q, ConnectApi.ZoneSearchResultType filter)
```

#### Parameters

***communityId***

Type: String

Use 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(String, String, String, ConnectApi.ZoneSearchResultType, ConnectApi.ZoneSearchPage)*
*Testing ConnectApi Code*

## searchInZone(String, String, String, ConnectApi.ZoneSearchResultType, String, Integer)

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

#### Signature

```
public static ConnectApi.ZoneSearchPage searchInZone(String communityId, String zoneId,
String q, ConnectApi.ZoneSearchResultType filter, String pageParam, Integer pageSize)
```

#### Parameters

***communityId***

Type: String

Use `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 items per page. Valid values are between 1 and 100. If you pass in `null`, the default size is 25.

#### Return Value

Type: `ConnectApi.ZoneSearchPage`

### See Also:

*setTestSearchInZone(String, String, String, ConnectApi.ZoneSearchResultType, String, Integer, ConnectApi.ZoneSearchPage)*
*Testing ConnectApi Code*

## setTestSearchInZone(String, String, String, ConnectApi.ZoneSearchResultType, ConnectApi.ZoneSearchPage)

Registers a `ConnectApi.ZoneSearchPage` object to be returned when `searchInZone(String, String, String, ConnectApi.ZoneSearchResultType)` 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 `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(String, String, String, ConnectApi.ZoneSearchResultType)*
*Testing ConnectApi Code*

## setTestSearchInZone(String, String, String, ConnectApi.ZoneSearchResultType, String, Integer, ConnectApi.ZoneSearchPage)

Registers a `ConnectApi.ZoneSearchPage` object to be returned when `searchInZone(String, String, String, ConnectApi.ZoneSearchResultType, String, Integer)` 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 `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 items per page. Valid values are between 1 and 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(String, String, String, ConnectApi.ZoneSearchResultType, String, Integer)*
*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 take no arguments.

Some methods take arguments typed with an abstract class. You must pass in an instance of a concrete child class for these arguments.

Most properties for the input classes can be set. Read-only properties are noted in this documentation.

### `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 |
| `filename` | String | File name with the file extension, such as UserPhoto.jpg | 28.0 |

### `ConnectApi.CanvasAttachmentInput` Class

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 |
| `developerName` | String | The developer name (API name) of the canvas app | 29.0 |
| `height` | String | Optional. The height of the canvas app in pixels. Default height is 200 pixels. | 29.0 |
| `namespacePrefix` | String | Optional. The namespace prefix of the Developer Edition organization in which the canvas app was created. | 29.0 |
| `parameters` | String | Optional. Parameters passed to the canvas app in JSON format. Example: `{'isUpdated'='true'}` | 29.0 |
| `thumbnailUrl` | String | Optional. A URL to a thumbnail image for the canvas app. Maximum dimensions are 120x120 pixels. | 29.0 |
| `title` | String | The title of the link used to call the canvas app. | 29.0 |

### `ConnectApi.ChatterGroupInput` Class

| Property | Type | Description | Available |
|---|---|---|---|
| `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. Group Information Input Class` | The “Information” section of a group. 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 |

| Property | Type | Description | Available |
|---|---|---|---|
| owner | String | The ID of the group owner. This property is available for PATCH requests only. | 29.0 |
| visibility | ConnectApi.GroupVisibilityType | Specifies whether a group is private or public.<br>• PrivateAccess—Only members of the group can see posts to this group.<br>• PublicAccess—All users within the internal community can see posts to this group. | 29.0 |

### 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:<br>• ContentAttachmentInput<br>• NewFileAttachmentInput<br>LinkAttachmentInput is not permitted for comments. | 28.0 |
| body | ConnectApi.MessageBodyInput Class | Description of message body. The body can contain up to 25 mentions. | 28.0 |

### ConnectApi.ContentAttachmentInput Class

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 |

### ConnectApi.FeedItemAttachmentInput 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 add rich feed items, for example, feed items that include @mentions or files. Also used to bookmark a feed item.

| Property | Type | Description | Available Version |
|---|---|---|---|
| attachment | ConnectApi.FeedItem AttachmentInput Class | Specifies the attachment for the feed item. The feed item type is inferred based on the provided attachment. | 28.0 |
| body | ConnectApi.MessageBody Input Class | Message body. The body can contain up to 25 mentions. | 28.0 |
| isBookmarked ByCurrentUser | Boolean | Specifies if the new feed item should be bookmarked for the user (true) or not (false). | 28.0 |
| originalFeedItemId | String | The 18-character ID of a feed item to share. | 28.0 |
| visibility | ConnectApi.FeedItem VisibilityType Enum | Specifies the type of feed item, such as a content post, a text post, and so on.<br>• AllUsers—Visibility is not limited to internal users.<br>• InternalUsers—Visibility is limited to internal users. | 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.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<br>  **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

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 |
| urlName | String | Title of the link | 28.0 |

### `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.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<br>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

### `ConnectApi.NewFileAttachmentInput` Class

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 |
| `title` | String | File's title | 28.0 |

### `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.<br>  **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

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 |

### `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 |

### `ConnectApi.TopicInput` Class

Used to update a topic's description and the spacing and capitalization of a topic's name.

| Property | Type | Description | Available Version |
|---|---|---|---|
| `description` | String | Description of the topic | 29.0 |
| `name` | String | Name of the topic | 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 Class>` | 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.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 attachments are typed as `ConnectApi.FeedItemAttachment`. 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:<br>• `Address`<br>• `Blank`<br>• `Boolean`<br>• `Compound`<br>• `CreatedBy`<br>• `Date`<br>• `DateTime`<br>• `Email`<br>• `LastModifiedBy`<br>• `Location`<br>• `Name`<br>• `Number`<br>• `Percent`<br>• `Phone`<br>• `Picklist`<br>• `Reference`<br>• `Text`<br>• `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.Actor` Class

Superclass of:

- `ConnectApi.ActorWithId Class`

| Name | Type | Description | Available Version |
|---|---|---|---|
| `name` | String | Name of the actor, such as the group name | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| type | String | One of the following:<br>• file<br>• group<br>• unauthenticateduser<br>• user<br>• *record type name*—the name of the record type, such as myCustomObject__c | 28.0 |

### ConnectApi.ActorWithId Class

Subclass of: ConnectApi.Actor Class

Superclass of:

- ConnectApi.AbstractRecordView Class on page 560
- ConnectApi.ChatterGroup Class
- ConnectApi.File Class
- ConnectApi.User

| 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 Class | If the context user is following the item, this contains information about the subscription, else returns null. | 28.0 |
| recordViewUrl | String | The URL of the Chatter REST API record view resource for this record, or null if this record does not have a record view layout, or if "Connect Records API" is not enabled for this organization, or if the record type isn't supported. | 30.0 |
| 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.BasicTemplateAttachment` Class

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 |
| `icon` | `ConnectApi.Icon Class` | An optional icon. | 28.0 |
| `linkRecordId` | String | If `linkURL` refers to a Database.com record, `linkRecordId` contains the ID of the record. | 28.0 |
| `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 |
| `title` | String | An optional title to the detail page. If `linkUrl` is specified, the title links to `linkUrl`. | 28.0 |

### `ConnectApi.BlankRecordField` Class

Subclass of `ConnectApi.AbstractRecordField Class`

A record field displayed as a place holder in a grid of fields.

### `ConnectApi.CanvasTemplateAttachment` Class

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 |
| `developerName` | String | Specifies the developer name (API name) of the canvas app. | 29.0 |
| `height` | String | Optional. The height of the canvas app in pixels. Default height is 200 pixels. | 29.0 |
| `icon` | `ConnectApi.Icon Class` | The canvas app icon. | 29.0 |
| `namespacePrefix` | String | Optional. The namespace prefix of the Developer Edition organization in which the canvas app was created. | 29.0 |
| `parameters` | String | Optional. Parameters passed to the canvas app in JSON format. Example: `{'isUpdated'='true'}` | 29.0 |
| `thumbnailUrl` | String | Optional. A URL to a thumbnail image for the canvas app. Maximum dimensions are 120x120 pixels. | 29.0 |
| `title` | String | Specifies the title of the link used to call the canvas app. | 29.0 |

### `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.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 |
| `canHaveChatterGuests` | Boolean | `true` if this group allows Chatter guests | 28.0 |
| `community` | `ConnectApi.Reference Class` | 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.<br>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 |
| `lastFeedItemPostDate` | Datetime | ISO8601 date string, for example, 2011-02-25T18:24:31.000Z of the most recent feed item posted to the group. | 28.0 |
| `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.<br>• `GroupOwner`<br>• `GroupManager` | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| | | • NotAMember<br>• NotAMemberPrivateRequested<br>• StandardMember | |
| mySubscription | ConnectApi.Reference Class | 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 Class | Information about the owner of the group | 28.0 |
| photo | ConnectApi.Photo Class | Information about the group photo | 28.0 |
| visibility | ConnectApi.GroupVisibilityType Enum | Specifies whether a group is private or public. Valid values are:<br>• PrivateAccess—Only members of the group can see posts to this group.<br>• PublicAccess—All users within the internal community can see posts to this group. | 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.Group Information Class | Describes the "Information" section of the group. 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.Chatter GroupDetail Class> | 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. | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| 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 Class> | 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. | 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 Class | A reference to the liked comment or feed item. | 28.0 |
| url | String | Like's Chatter REST API URL | 28.0 |
| user | ConnectApi.UserSummary Class | 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 |
| likes | List<ConnectApi.ChatterLike Class> | List of likes | 28.0 |
| nextPageToken | Integer | Token identifying the next page or null if there isn't a next page. | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| nextPageUrl | String | Chatter REST API URL identifying the next page or null if there isn't a next page. | 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.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.FeedItem Attachment Class | If the comment contains an attachment, property value is ContentAttachment. If the comment does not contain an attachment, it is null. | 28.0 |
| body | ConnectApi.FeedBody Class | Body of the comment | 28.0 |
| clientInfo | ConnectApi. ClientInfo Class | 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 |
| feedItem | ConnectApi.Reference Class | Information about the feed item | 28.0 |
| id | String | Comment's 18–character ID | 28.0 |
| isDelete Restricted | 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.Chatter LikePage Class | The first page of likes for the comment | 28.0 |
| likesMessage | ConnectApi.Message Body Class | A message body that describes who likes the comment. | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| moderationFlags | ConnectApi.ModerationFlags Class | Information about the moderation flags on a comment. If ConnectApi.Features.communityModeration is false, this property is null. | 29.0 |
| myLike | ConnectApi.Reference Class | If the context user has liked the comment, this property is a reference to the specific like, null otherwise | 28.0 |
| parent | ConnectApi.Reference Class | 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 |
| type | ConnectApi.CommentType Enum | Specifies the type of comment.<br>• ContentComment—Comment contains an attachment.<br>• TextComment—Comment contains only text. | 28.0 |
| url | String | Chatter REST API URL to this comment | 28.0 |
| user | ConnectApi.User Summary Class | Information about the comment author | 28.0 |

## ConnectApi.CommentPage Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| comments | List<ConnectApi.Comment Class> | Collection of comments | 28.0 |
| currentPageToken | String | Token identifying the current page. | 28.0 |
| currentPageUrl | String | Chatter REST API URL identifying the current page. | 28.0 |
| 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. | 28.0 |
| total | Integer | Total number of comments for the parent feed item | 28.0 |

## ConnectApi.Community Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| 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 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| knowledgeableEnabled | Boolean | Specifies whether knowledgeable people and endorsements are available for topics (true), or not (false). | 30.0 |
| name | String | Community name | 28.0 |
| 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 |
| sendWelcomeEmail | Boolean | Send email to all new users when they join | 28.0 |
| siteUrl | String | Site URL for the community, which is the custom domain plus a URL prefix | 30.0 |
| status | ConnectApi.CommunityStatus Enum | Specifies the current status of the community.<br>• Live<br>• Inactive<br>• 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 Class> | 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 Class> | List of message segments. | 28.0 |

### ConnectApi.CompoundRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field that is a composite of sub-fields.

| Name | Type | Description | Available Version |
|---|---|---|---|
| fields | List<ConnectApi.AbstractRecordField Class> | A collection of sub-fields that make up the compound field. | 29.0 |

### ConnectApi.ContentAttachment Class

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 |
| description | String | Description of the attachment | 28.0 |
| downloadUrl | String | File's URL | 28.0 |
| fileExtension | String | File's extension | 28.0 |
| fileSize | String | Size of the file in bytes. If size cannot be determined, returns unknown. | 28.0 |
| fileType | String | Type of file | 28.0 |
| 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 |
| id | String | Content's 18-character ID | 28.0 |
| isInMyFileSync | Boolean | true if the file is included in the user's Salesforce Files folder, and is synced between that folder and Chatter; false otherwise. | 28.0 |
| mimeType | String | File's MIME type | 28.0 |
| renditionUrl | String | URL to the file's rendition resource | 28.0 |
| renditionUrl240By180 | String | URL to the 240 x 180 rendition resource for the file. Renditions are processed asynchronously and might not be available immediately after the file has been uploaded. | 30.0 |
| renditionUrl720By480 | String | URL to the 720 x 480 rendition resource for the file. Renditions are processed asynchronously and might not be available immediately after the file has been uploaded. | 30.0 |
| textPreview | String | Text preview of the file if available, null otherwise. | 30.0 |
| thumb120By90RenditionStatus | String | Specifies the rendering status of the 120 x 90 preview image of the file. One of these values:<br>• Processing—Image is being rendered.<br>• Failed—Rendering process failed.<br>• Success—Rendering process was successful.<br>• Na—Rendering is not available for this image. | 30.0 |
| thumb240By180RenditionStatus | String | Specifies the rendering status of the 240 x 180 preview image of the file. One of these values:<br>• Processing—Image is being rendered.<br>• Failed—Rendering process failed. | 30.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| | | • `Success`—Rendering process was successful.<br>• `Na`—Rendering is not available for this image. | |
| `thumb720By480RenditionStatus` | String | Specifies the rendering status of the 720 x 480 preview image of the file. One of these values:<br>• `Processing`—Image is being rendered.<br>• `Failed`—Rendering process failed.<br>• `Success`—Rendering process was successful.<br>• `Na`—Rendering is not available for this image. | 30.0 |
| `title` | String | Title of the file | 28.0 |
| `versionId` | String | File's version number | 28.0 |

### `ConnectApi.CurrencyRecordField` Class

Subclass of `ConnectApi.LabeledRecordField Class`

A record field containing a currency value.

### `ConnectApi.DateRecordField` Class

Subclass of `ConnectApi.LabeledRecordField Class`

A record field containing a date.

| Name | Type | Description | Available Version |
|---|---|---|---|
| `dateValue` | Date | A date that a machine can read.<br>Ignore the trailing `00:00:00.000Z` characters. | 29.0 |

### `ConnectApi.EntityLinkSegment` Class

Subclass of `ConnectApi.MessageSegment Class`

| Name | Type | Description | Available Version |
|---|---|---|---|
| `motif` | `ConnectApi.Motif` | 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 Class` | A reference to the link object if applicable, otherwise, `null`. | 28.0 |

### `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 |

| Property | Type | Description | Available Version |
|---|---|---|---|
| chatterAnswers | Boolean | Indicates whether Chatter Answers is enabled | 29.0 |
| chatterGlobalInfluence | Boolean | Indicates whether the user details include global Chatter activity | 28.0 |
| chatterGroupRecords | Boolean | Reserved for future use | 30.0 |
| chatterGroupRecordSharing | Boolean | Reserved for future use | 30.0 |
| chatterMessages | Boolean | Indicates whether Chatter messages are enabled for the organization | 28.0 |
| chatterTopics | Boolean | Indicates whether Chatter topics are enabled | 28.0 |
| communityModeration | Boolean | Specifies whether Community moderation is enabled. | 29.0 |
| connectRecords | Boolean | Reserved for future use | 28.0 |
| dashboardComponentSnapshots | Boolean | Indicates whether the user can post dashboard component snapshots | 28.0 |
| defaultCurrencyIsoCode | String | The ISO code of the default currency. Applicable only when `multiCurrency` is false. | 28.0 |
| feedPolling | Boolean | Indicates whether the is-modified resource is enabled for the Chatter API | 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 |
| 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 publisher actions are enabled | 28.0 |
| 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 |
|---|---|---|---|
| feedItemsUrl | String | Chatter REST API URL of feed items | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| 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 is not 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.

As of API version 29.0, the value of the ConnectApi.FeedItem.FeedBody.text property can be null, which means you can't use it as the default case for rendering text. Instead, use the ConnectApi.FeedItem.MessageBody.text property as the default case.

### ConnectApi.FeedDirectory Class

A directory of feeds and favorites.

| Name | Type | Description | Available Version |
|---|---|---|---|
| favorites | List<ConnectApi.FeedFavorite Class> | A list of feed favorites | 30.0 |
| feeds | List<ConnectApi.FeedDirectoryItem Class> | A list of feeds | 30.0 |

### ConnectApi.FeedDirectoryItem Class

The definition of a feed.

| Name | Type | Description | Available Version |
|---|---|---|---|
| feedItemsUrl | String | Chatter REST API resource URL for the feed items of a specific feed. | 30.0 |
| feedType | ConnectApi .FeedType | The feed type. One of these values:<br>• Bookmarks—Contains all feed items saved as bookmarks by the logged-in user.<br>• 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.<br>• Files—Contains all feed items that contain files posted by people or groups that the logged-in user follows.<br>• Filter—Contains the news feed filtered to contain feed items whose parent is a specified object type.<br>• Groups—Contains all feed items from all groups the logged-in user either owns or is a member of.<br>• 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. | 30.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| | | • `News`—Contains all updates for people the logged-in user follows, groups the user is a member of, files and records the user is following, all updates for records whose parent is the logged-in user, and every feed item and comment that mentions the logged-in user or that mentions a group the logged-in user is a member of.<br>• `People`—Contains all feed items posted by all people the logged-in user follows.<br>• `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.<br>• `To`—Contains all feed items with mentions of the logged-in user, feed items the logged-in user commented on, and feed items created by the logged-in user that are commented on.<br>• `Topics`—Contains all feed items that include the specified topic.<br>• `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 | 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`.<br><br>A *key prefix* is the first three characters of a record ID, which specifies the entity type. | 30.0 |
| `label` | String | Localized label of the feed | 30.0 |

## `ConnectApi.FeedFavorite` Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| `community` | `ConnectApi.Reference Class` | Information about the community that contains the favorite | 28.0 |
| `createdBy` | `ConnectApi.UserSummary Class` | 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 Class | A reference to the topic if applicable, null otherwise | 28.0 |
| type | ConnectApi.FeedFavoriteType Enum | An empty string or one of the following values:<br>• ListView<br>• Search<br>• Topic | 28.0 |
| url | String | Chatter REST API URL to this favorite | 28.0 |
| user | ConnectApi.UserSummary Class | Information about the user who saved this favorite | 28.0 |

### ConnectApi.FeedFavorites Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| favorites | List<ConnectApi.FeedFavorite Class> | Complete list of favorites | 28.0 |
| total | Integer | Total number of favorites | 28.0 |

### ConnectApi.FeedItem Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| actor | ConnectApi.Actor Class | The entity that created the feed item. | 28.0 |
| attachment | ConnectApi.FeedItemAttachment Class | Information about the attachment. If there is no attachment, returns null. | 28.0 |
| body | ConnectApi.FeedBody Class | The body of the feed item.<br>As of API version 29.0, the value of the ConnectApi.FeedItem.body.text property can be null, which means you can't use it as the default case for rendering text. Instead, use the ConnectApi.FeedItem.preamble.text property as the default case. | 28.0 |
| canShare | Boolean | true if the feed item can be shared, otherwise, false | 28.0 |
| clientInfo | ConnectApi.ClientInfo Class | Information about the Connected App used to authenticate the connection. | 28.0 |
| comments | ConnectApi.CommentPage Class | First page of comments for this feed item. | 28.0 |
| createdDate | Datetime | ISO8601 date string, for example, 2011-02-25T18:24:31.000Z | 28.0 |
| id | String | Feed item's 18–character ID | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| isBookmarkedByCurrentUser | Boolean | true if the current user has bookmarked this feed item, otherwise, false. | 28.0 |
| 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 |
| isEvent | Boolean | true if feed item created due to an event change, otherwise, false | 28.0 |
| isLikedByCurrentUser | Boolean | true if the current user has liked this feed item, otherwise, false | 28.0 |
| likes | ConnectApi.ChatterLikePage Class | First page of likes for this feed item | 28.0 |
| likesMessage | ConnectApi.MessageBody Class | A message body the describes who likes the feed item. | 28.0 |
| moderationFlags | ConnectApi.ModerationFlags Class | Information about the moderation flags on a feed item. If ConnectApi.Features.communityModeration is false, this property is null. | 29.0 |
| modifiedDate | Datetime | ISO8601 date string, for example, 2011-02-25T18:24:31.000Z | 28.0 |
| myLike | ConnectApi.Reference Class | If the context user has liked the feed item, this property is a reference to the specific like, otherwise, null. | 28.0 |
| originalFeedItem | ConnectApi.Reference Class | A reference to the original feed item if this feed item is a shared feed item, otherwise, null. | 28.0 |
| originalFeedItemActor | ConnectApi.Actor Class | If this feed item is a shared feed item, returns information about the original poster of the feed item, otherwise, returns null. | 28.0 |
| parent | ConnectApi.ActorWithId Class | Feed item's parent | 28.0 |
| photoUrl | String | URL of the photo associated with the feed item | 28.0 |
| preamble | ConnectApi.MessageBody Class | 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.<br><br>As of API version 29.0, the value of the ConnectApi.FeedItem.body.text property can be null, which means you can't use it as the default case for rendering text. Instead, use the ConnectApi.FeedItem.preamble.text property as the default case. | 28.0 |
| topics | ConnectApi.FeedItemTopicPage Class | Topics for this feed item | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| `type` | `ConnectApi.FeedItemType` Enum | Specifies the type of feed item, such as a content post, a text post, and so on.<br>• `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.<br>• `ApprovalPost`—Feed item with an approval action attachment. Approvers can act on the feed item parent.<br>• `AttachArticleEvent`—Feed item generated when an article is attached to a case in Case Feed.<br>• `BasicTemplateFeedItem`—Feed item with a standard rendering containing an attachment with an image, link, and title.<br>• `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.<br>• `CollaborationGroupCreated`—Feed item generated when a new group is created. Contains a link to the new group.<br>• `CollaborationGroupUnarchived`—Do not use. Feed item generated when an archived group is activated.<br>• `ContentPost`—Feed item with a file attachment.<br>• `LinkPost`—Feed item with a hyperlink attachment<br>• `PollPost`—Feed item with an actionable poll attachment. Viewers of the feed item are allowed to vote on the options in the poll.<br>• `ReplyPost`—Feed item generated by a Chatter Answers reply.<br>• `RypplePost`—Feed item generated when a Thanks badge is created.<br>• `TextPost`—Feed item without an attachment.<br>• `TrackedChange`—Feed item created when one or more fields on a record have been changed.<br>• `SocialPost`—Feed item generated when a social post is created from a case in Case Feed.<br>• `UserStatus`—<br>Deprecated. A user's post to their own profile. | 28.0 |
| `url` | String | Chatter REST API URL to this feed item | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| visibility | ConnectApi.FeedItemVisibilityType Enum | Specifies the type of users who can see a feed item.<br>• AllUsers—Visibility is not limited to internal users.<br>• InternalUsers—Visibility is limited to internal users. | 28.0 |

### ConnectApi.FeedItemAttachment Class

This class is abstract.

Subclasses:

- ConnectApi.BasicTemplateAttachment Class
- ConnectApi.CanvasTemplateAttachment Class
- ConnectApi.ContentAttachment 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 attachments are typed as ConnectApi.FeedItemAttachment. 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

A paged collection of ConnectApi.FeedItem objects.

| Name | Type | Description | Available Version |
|---|---|---|---|
| currentPageToken | String | Token identifying the current page. | 28.0 |
| currentPageUrl | String | Chatter REST API URL identifying the current page. | 28.0 |
| isModifiedToken | String | An opaque polling token to use in the *since* parameter of the ChatterFeeds.isModified method. The token describes when the feed was last modified. | 28.0 |
| isModifiedUrl | 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 is not a news feed. Use this URL to poll a news feed for updates. | 28.0 |
| items | List<ConnectApi.FeedItem Class> | List of feed items | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| 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. | 28.0 |
| updatesToken | String | Token to use in an updatedSince parameter, or null if not available. | 30.0 |
| 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. Property is null if not available. | 30.0 |

### ConnectApi.FeedItemTopicPage Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| canAssignTopics | Boolean | true if a topic can be assigned to the feed item, false otherwise | 28.0 |
| topics | List<ConnectApi.Topic Class> | List of topics | 28.0 |

### ConnectApi.FeedModifiedInfo Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| isModified | Boolean | true if the news feed has been modified since the last time it was polled; false otherwise. Returns null if the feed is not a news feed. | 28.0 |
| isModifiedToken | String | An opaque polling token to use in the *since* parameter of the ChatterFeeds.isModified method. The token describes when the feed was last modified. | 28.0 |
| nextPollUrl | 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 is not a news feed. Use this URL to poll a news feed for updates. Make a request to this URL after requesting the URL in the Feed.isModifiedUrl or FeedItemPage.isModifiedUrl property. | 28.0 |

### ConnectApi.FeedPoll Class

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.FeedPoll | List of choices for poll. | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| | Choice Class> | | |
| 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 |
| totalVoteCount | Integer | Total number of votes cast on the feed poll item. | 28.0 |

### 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 | Specifies the value type of a field change:<br>• NewValue—A new value<br>• 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 |
| 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 included in the user's Salesforce Files folder, and is synced between that folder and Chatter; `false` otherwise. | 28.0 |
| mimeType | String | File's MIME type | 28.0 |
| moderationFlags | ConnectApi.ModerationFlags Class | Information about the moderation flags on a file. If `ConnectApi.Features.communityModeration` is `false`, this property is `null`. | 30.0 |
| modifiedDate | Datetime | ISO8601 date string, for example, 2011-02-25T18:24:31.000Z | 28.0 |
| name | String | Name of the file | 28.0 |
| origin | String | Specifies the file source. Valid values are:<br>• `Chatter`—file came from Chatter<br>• `Content`—file came from content | 28.0 |
| owner | ConnectApi.User Summary Class | File's owner | 28.0 |
| pdfRenditionStatus | String | Specifies if a PDF preview version of the file has been rendered | 28.0 |
| publishStatus | ConnectApi.FilePublishStatus | Specifies the publish status of the file. | 28.0 |
| renditionUrl | String | URL to the rendition for the file | 28.0 |
| renditionUrl240By180 | String | URL to the 240 x 180 rendition resource for the file. Renditions are processed asynchronously and might not be available immediately after the file has been uploaded. | 29.0 |
| renditionUrl720By480 | String | URL to the 720 x 480 rendition resource for the file. Renditions are processed asynchronously and might not be available immediately after the file has been uploaded. | 29.0 |
| sharingRole | ConnectApi.FileSharingType | Specifies the sharing role of the file. | 28.0 |
| textPreview | String | Text preview of the file if available, `null` otherwise. | 30.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| `thumb120By90 RenditionStatus` | String | Specifies the rendering status of the 120 x 90 preview image of the file. One of these values:<br>• `Processing`—Image is being rendered.<br>• `Failed`—Rendering process failed.<br>• `Success`—Rendering process was successful.<br>• `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:<br>• `Processing`—Image is being rendered.<br>• `Failed`—Rendering process failed.<br>• `Success`—Rendering process was successful.<br>• `Na`—Rendering is not available for this image. | 28.0 |
| `thumb720By480 RenditionStatus` | String | Specifies the rendering status of the 720 x 480 preview image of the file. One of these values:<br>• `Processing`—Image is being rendered.<br>• `Failed`—Rendering process failed.<br>• `Success`—Rendering process was successful.<br>• `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 Class>` | List of subscriptions | 28.0 |
| `nextPageUrl` | String | Chatter REST API URL identifying the next page or `null` if there isn't a next 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 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<br>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 Class>` | List of subscriptions | 28.0 |
| `nextPageUrl` | String | Chatter REST API URL identifying the next page or `null` if there isn't a next 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 records being followed across all pages | 28.0 |

### `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. 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 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| 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 |
| role | ConnectApi.GroupMembershipType Enum | Specifies the type of membership the user has with the group, such as group owner, manager, or member.<br>• GroupOwner<br>• GroupManager<br>• NotAMember<br>• NotAMemberPrivateRequested<br>• StandardMember | 28.0 |
| url | String | Chatter REST API URL to this membership | 28.0 |
| user | ConnectApi.User Summary Class | 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 Class> | List of group members | 28.0 |
| myMembership | ConnectApi.Reference Class | 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. | 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 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| requestedGroup | ConnectApi.Reference Class | 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.<br>The maximum length is 756 characters. | 28.0 |
| status | ConnectApi.GroupMembershipRequestStatus Enum | The status of a request to join a private group. Values are:<br>• Accepted<br>• Declined<br>• Pending | 28.0 |
| url | String | URL of the group membership request object. | 28.0 |
| user | ConnectApi.UserSummary Class | Information about the user requesting membership in a group. | 28.0 |

### ConnectApi.GroupMembershipRequests Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| requests | List<ConnectApi.GroupMembershipRequest Class> | Information about group membership requests. | 28.0 |
| total | Integer | The total number of requests. | 28.0 |

### 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:<br>`/services/data/v30.0/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:<br>`/services/data/v30.0/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 |

| Property | Type | Description | Available Version |
|---|---|---|---|
| `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 |
| `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 and if it isn't recognized, render the text value as the default case. | 29.0 |

### `ConnectApi.LinkAttachment` Class

Subclass of ConnectApi.FeedItemAttachment Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| `title` | String | Title given to the link if available, otherwise, `null` | 28.0 |
| `url` | String | The link URL | 28.0 |

### `ConnectApi.LinkSegment` Class

Subclass of ConnectApi.MessageSegment Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| `url` | String | The link URL | 28.0 |

### `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 | 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`.<br>One of these values:<br>• `ChatterGuest`—User is a Chatter customer in an external group<br>• `ChatterOnly`—User is a Chatter Free customer<br>• `Guest`—Unauthenticated users<br>• `Internal`—User is a standard organization member<br>• `Portal`—User is a Customer Portal User, a communities user, and so on.<br>• `System`—User is Chatter Expert or a system user<br>• `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 Class>` | 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. | 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 Class | Information about the mentioned user or group | 29.0 |
| user | ConnectApi.UserSummary Class | Information about the mentioned user | 28.0 only<br>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 |
| validationStatus | | Specifies the type of validation error for a proposed mention, if any.<br>• Disallowed—The proposed mention is invalid and will be 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.<br>• Inaccessible—The proposed mention is allowed but the user or record being mentioned will not be notified because they don't have access to the parent record being discussed.<br>• 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, 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, hasErrors would be true and the group would | 29.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| | | have a `validationStatus` of `Disallowed` in its mention validation. | |
| `mentionValidations` | `List<ConnectApi.MentionValidation Class>` | 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 attachments are typed as `ConnectApi.FeedItemAttachment`. 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 | Formatted text of the message. | 28.0 |
| `type` | `ConnectApi.MessageSegmentType` Enum | Textual summary of the message segment. The values correspond to the different child classes. Value are:<br>• `ConnectApi.EntityLinkSegment Class`<br>• `ConnectApi.FieldChangeSegment Class`<br>• `ConnectApi.FieldChangeNameSegment Class`<br>• `ConnectApi.FieldChangeValueSegment Class`<br>• `ConnectApi.HashtagSegment Class`<br>• `ConnectApi.LinkSegment Class`<br>• `ConnectApi.MentionSegment Class` | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| | | • ConnectApi.MoreChangesSegment Class<br>• ConnectApi.ResourceLinkSegment Class<br>• ConnectApi.TextSegment Class | |

### 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 logged-in user is not a community moderator, the property is null. | 29.0 |
| flaggedByMe | Boolean | true if the logged-in 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 Class> | Complete list of tracked changes. | 29.0 |
| moreChangesCount | Integer | Number of additional changes | 28.0 |

### ConnectApi.Motif

The motif properties contain URLs for small, medium, and large icons that indicate the Database.com 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"
},
```

Users use the following icons:

```
  "motif": {
     "color: "1797C0",
     "largeIconUrl": "/img/icon/profile64.png",
     "mediumIconUrl": "/img/icon/profile32.png",
     "smallIconUrl": "/img/icon/profile16.png"
  },
```

Groups use the following icons:

```
  "motif": {
     "color: "1797C0",
     "largeIconUrl": "/img/icon/groups64.png",
     "mediumIconUrl": "/img/icon/groups32.png",
     "smallIconUrl": "/img/icon/groups16.png"
  },
```

Files use the following icons:

```
 "motif": {
     "color: "1797C0",
     "largeIconUrl": "/img/content/content64.png",
     "mediumIconUrl": "/img/content/content32.png",
     "smallIconUrl": "/img/icon/files16.png"
  },
```

**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 |

### `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 Class` | Information about features available in the organization | 28.0 |
| `name` | String | Organization name | 28.0 |
| `orgId` | String | 18-character ID for the organization | 28.0 |
| `userSettings` | `ConnectApi.UserSettings Class` | Information about the organization permissions for the user | 28.0 |

### `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 |
| `phoneNumber` | String | Phone number | 28.0 |
| `phoneType` | String | Phone type. Values are:<br>• `Fax`<br>• `Mobile`<br>• `Work`<br>These values are not localized. | 30.0 |
| `type` | String |  **Note:** This property is not available after version 29.0. Use the `phoneType` property instead.<br>Values are:<br>• `Fax`<br>• `Mobile`<br>• `Work`<br>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 |
| `standardEmail PhotoUrl` | String | A temporary URL to the small profile. The URL expires after 30 days and is available to unauthenticated users. | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| url | String | A resource that returns a Photo object: for example, /services/data/v30.0/chatter/users/005D0000001LL8OIAW/photo. | 28.0 |

### ConnectApi.PicklistRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field containing an enumerated value.

### ConnectApi.RecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A generic record field containing a label and text value.

### ConnectApi.RecordSnapshotAttachment Class

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 Class | The representation of the record. | 29.0 |

### 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 Class> | 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 Class> | 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 | The order of the fields to use in the fields property to lay out the fields in a record section.<br>• LeftRight—Fields are rendered from left to right.<br>• TopDown—Fields are rendered from the top down. | 29.0 |
| fields | ConnectApi.AbstractRecordField Class | 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 | 18–character id of the object being referenced | 28.0 |
| url | String | URL to the resource | 28.0 |

### ConnectApi.ReferenceRecordField Class

Subclass of ConnectApi.LabeledRecordField Class

A record field containing a reference to another object.

| Name | Type | Description | Available Version |
|---|---|---|---|
| reference | ConnectApi.RecordSummary Class | 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 | Date | A time at which the referenced object acted. | 29.0 |
| reference | ConnectApi.RecordSummary Class | The object referenced by the record field. | 29.0 |

### 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 Class | Information about the community in which the subscription exists | 28.0 |
| id | String | Subscription's 18–character ID | 28.0 |
| subject | ConnectApi.Actor Class | Information about the parent, that is, the thing or person being followed | 28.0 |
| subscriber | ConnectApi.Actor Class | 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 Database.com. 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 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| 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.User Summary Class | User being endorsed | 30.0 |
| endorsementId | String | 18-character ID of the endorsement record | 30.0 |
| endorser | ConnectApi.User Summary Class | User performing the endorsement | 30.0 |
| topic | ConnectApi.Topic Class | 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. | 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 Class> | List of topic endorsements | 30.0 |

### 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. | 29.0 |
| topics | List<ConnectApi.Topic Class> | List of topics | 29.0 |

### ConnectApi.TopicSuggestion Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| existingTopic | ConnectApi.Topic Class | 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 |
|---|---|---|---|
| TopicSuggestionPage | List<ConnectApi.TopicSuggestion Class> | List of topic suggestions | 29.0 |

### ConnectApi.TrackedChangeAttachment Class

| Name | Type | Description | Available Version |
|---|---|---|---|
| changes | List<ConnectApi.TrackedChangeItem Class> | A list of tracked changes. | 28.0 |

### 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.User

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 |
| companyName | String | Name of the company | 28.0 |
| firstName | String | User's first name | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| 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 Class | Information about the user's photos | 28.0 |
| title | String | User's title | 28.0 |
| userType | ConnectApi.UserType Enum | Specifies the type of user.<br>• ChatterGuest—User is a Chatter customer in an external group<br>• ChatterOnly—User is a Chatter Free customer<br>• Guest—Unauthenticated users<br>• Internal—User is a standard organization member<br>• Portal—User is a Customer Portal User, a communities user, and so on.<br>• System—User is Chatter Expert or a system user<br>• Undefined—User is a user type that is a custom object. | 28.0 |

### ConnectApi.UserCapabilities

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

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 Class | User's address | 28.0 |
| chatterActivity | ConnectApi.ChatterActivity Class | Chatter activity statistics | 28.0 |
| chatterInfluence | ConnectApi.GlobalInfluence Class | 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 Class | Information about items the user is following | 28.0 |
| groupCount | Integer | Number of groups user is following | 28.0 |
| 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 |
| thanksReceived | Integer | The number of times the user has been thanked. | 29.0 |
| phoneNumbers | List<ConnectApi.PhoneNumber Class> | Collection of user's phone numbers | 28.0 |
| 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 Class> | List of groups | 28.0 |

| Name | Type | Description | Available Version |
|---|---|---|---|
| nextPageUrl | String | Chatter REST API URL identifying the next page or null if there isn't a next page. | 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 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. | 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.User Detail Class> | 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

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 |
| userDetail | ConnectApi.UserDetail | The details about the user attached to the profile | 29.0 |

### ConnectApi.UserProfileTab

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 | Specifies the type of tab | 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 |
| 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 |
| fileSyncStorageLimit | Integer | Maximum storage for synced files, in megabytes (MB) | 29.0 |
| hasAccessToInternalOrg | Boolean | User is a member of the internal organization | 28.0 |
| hasFileSync | Boolean | User has "Sync Files" permission. | 28.0 |
| hasRestDataApiAccess | Boolean | User has access to REST API. | 29.0 |
| timeZone | ConnectApi.TimeZone Class | The user's time zone as selected in My Settings in Database.com. 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 |

| Property | Type | Description | Available Version |
|---|---|---|---|
| `userLocale` | String | Locale of user | 28.0 |

### `ConnectApi.UserSummary` Class

Subclass of `ConnectApi.User`

| Name | Type | Description | Available Version |
|---|---|---|---|
| `isActive` | Boolean | `true` if user is active; `false` otherwise | 28.0 |

### `ConnectApi.Zone`

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<br>• `Community`—Available in a community.<br>• `Internal`—Available internally only.<br>• `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`

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 | Token identifying the next page or `null` if there isn't a next page. | 29.0 |

### ConnectApi.ZoneSearchPage

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. | 29.0 |

### ConnectApi.ZoneSearchResult

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 | Specifies the zone search result type<br>• Article—Search results contain only articles.<br>• 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 ConnectApi.

ConnectApi enums inherit all properties and methods of Apex enums.

| Enum | Description |
|---|---|
| ConnectApi.CommentType | Specifies the type of comment.<br>• ContentComment—Comment contains an attachment.<br>• TextComment—Comment contains only text. |
| ConnectApi.CommunityFlagVisibility | Specifies the visibility behavior of a flag for various user types.<br>• ModeratorsOnly—The flag is visible only to users with moderation permissions on the flagged item. |

| Enum | Description |
| --- | --- |
| | • `SelfAndModerators`—The flag is visible to the creator of the flag and to users with moderation permissions on the flagged item. |
| `ConnectApi.CommunityStatus` | Specifies the current status of the community.<br>• `Live`<br>• `Inactive`<br>• `UnderConstruction` |
| `ConnectApi.FeedDensity` | Specifies the density of the feed.<br>• `AllUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of.<br>• `FewerUpdates`—Displays all posts and comments from people and records the user follows and groups the user is a member of, but hides system-generated posts from records that nobody commented on. |
| `ConnectApi.FeedFavoriteType` | Specifies the origin of the feed favorite, such as whether it's a search term or a list view:<br>• `ListView`<br>• `Search`<br>• `Topic` |
| `ConnectApi.FeedItemAttachmentInputType` | Specifies the attachment type for feed item input objects:<br>• `Canvas`—A feed item that contains the metadata to create a canvas app.<br>• `ExistingContent`—A feed item with a previously uploaded file attached.<br>• `Link`—A feed item with a URL attached.<br>• `NewFile`—A feed item with a file attachment that was uploaded with the post.<br>• `Poll`—A feed item with a poll attached. |
| `ConnectApi.FeedItemAttachmentType` | Specifies the attachment type for feed item output objects:<br>• `Approval`—A feed item requiring approval.<br>• `BasicTemplate`—A feed item with a generic rendering of an image, link, and title.<br>• `Canvas`—A feed item that contains the metadata to render a link to a canvas app.<br>• `CaseComment`—A feed item created from a comment to a case record.<br>• `Content`—A feed item with a file attached.<br>• `DashboardComponent`—A feed item with a dashboard attached.<br>• `EmailMessage`—An email attached to a case record in Case Feed.<br>• `Link`—A feed item with a URL attached.<br>• `Poll`—A feed item with a poll attached. |

| Enum | Description |
| --- | --- |
| | • `RecordSnapshot`—The feed item attachment contains a view of a record at a single `ConnectApi.FeedItemType.CreateRecordEvent`.<br>• `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, a text post, and so on.<br>• `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.<br>• `ApprovalPost`—Feed item with an approval action attachment. Approvers can act on the feed item parent.<br>• `AttachArticleEvent`—Feed item generated when an article is attached to a case in Case Feed.<br>• `BasicTemplateFeedItem`—Feed item with a standard rendering containing an attachment with an image, link, and title.<br>• `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.<br>• `CollaborationGroupCreated`—Feed item generated when a new group is created. Contains a link to the new group.<br>• `CollaborationGroupUnarchived`—Do not use. Feed item generated when an archived group is activated.<br>• `ContentPost`—Feed item with a file attachment.<br>• `LinkPost`—Feed item with a hyperlink attachment<br>• `PollPost`—Feed item with an actionable poll attachment. Viewers of the feed item are allowed to vote on the options in the poll.<br>• `ReplyPost`—Feed item generated by a Chatter Answers reply.<br>• `RypplePost`—Feed item generated when a Thanks badge is created.<br>• `TextPost`—Feed item without an attachment.<br>• `TrackedChange`—Feed item created when one or more fields on a record have been changed.<br>• `SocialPost`—Feed item generated when a social post is created from a case in Case Feed.<br>• `UserStatus`—<br>Deprecated. A user's post to their own profile. |
| `ConnectApi.FeedItemVisibilityType` | Specifies the type of users who can see a feed item.<br>• `AllUsers`—Visibility is not limited to internal users.<br>• `InternalUsers`—Visibility is limited to internal users. |

| Enum | Description |
| --- | --- |
| `ConnectApi.FeedSortOrder` | Specifies the order returned by the sort, such as by date created or last modified:<br>• `CreatedDateDesc`—Sorts the feed items by most recent post date.<br>• `LastModifiedDateDesc`—Sorts the feed items by most recent activity, which includes new feed items and comments. |
| `ConnectApi.FeedType` | Specifies the type of feed:<br>• `Bookmarks`—Contains all feed items saved as bookmarks by the logged-in user.<br>• `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.<br>• `Files`—Contains all feed items that contain files posted by people or groups that the logged-in user follows.<br>• `Filter`—Contains the news feed filtered to contain feed items whose parent is a specified object type.<br>• `Groups`—Contains all feed items from all groups the logged-in user either owns or is a member of.<br>• `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.<br>• `News`—Contains all updates for people the logged-in user follows, groups the user is a member of, files and records the user is following, all updates for records whose parent is the logged-in user, and every feed item and comment that mentions the logged-in user or that mentions a group the logged-in user is a member of.<br>• `People`—Contains all feed items posted by all people the logged-in user follows.<br>• `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.<br>• `To`—Contains all feed items with mentions of the logged-in user, feed items the logged-in user commented on, and feed items created by the logged-in user that are commented on.<br>• `Topics`—Contains all feed items that include the specified topic.<br>• `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.FieldChangeValueType` | Specifies the value type of a field change:<br>• `NewValue`—A new value |

| Enum | Description |
| --- | --- |
| | • `OldValue`—An old value |
| `ConnectApi.FilePublishStatus` | The publish status of the file:<br>• `PendingAccess`—File is pending publishing.<br>• `PrivateAccess`—File is private.<br>• `PublicAccess`—File is public. |
| `ConnectApi.FileSharingType` | Specifies the sharing role of the file:<br>• `Admin`—Owner permission, but doesn't own the file.<br>• `Collaborator`—Viewer permission, and can edit, change permissions, and upload a new version of a file.<br>• `Owner`—Collaborator permission, and can make a file private, and delete a file.<br>• `Viewer`—Can view, download, and share a file.<br>• `WorkspaceManaged`—Permission controlled by the library. |
| `ConnectApi.GroupArchiveStatus` | Specifies a set of groups based on whether the groups are archived or not.<br>• `All`—All groups, including groups that are archived and groups that are not archived.<br>• `Archived`—Only groups that are archived.<br>• `NotArchived`—Only groups that are not archived. |
| `ConnectApi.GroupEmailFrequency` | Specifies the frequency with which a user receives email from a group.<br>• `EachPost`<br>• `DailyDigest`<br>• `WeeklyDigest`<br>• `Never`<br>• `UseDefault` |
| `ConnectApi.GroupMembershipType` | Specifies the type of membership the user has with the group, such as group owner, manager, or member.<br>• `GroupOwner`<br>• `GroupManager`<br>• `NotAMember`<br>• `NotAMemberPrivateRequested`<br>• `StandardMember` |
| `ConnectApi.GroupMembershipRequestStatus` | The status of a request to join a private group.<br>• `Accepted`<br>• `Declined`<br>• `Pending` |
| `ConnectApi.GroupVisibilityType` | Specifies whether a group is private or public.<br>• `PrivateAccess`—Only members of the group can see posts to this group. |

| Enum | Description |
| --- | --- |
| | • `PublicAccess`—All users within the internal community can see posts to this group. |
| `ConnectApi.MentionCompletionType` | Specifies the type of mention completion:<br>• `All`—All mention completions, regardless of the type of record to which the mention refers.<br>• `Group`—Mention completions for groups.<br>• `User`—Mention completions for users. |
| `ConnectApi.MentionValidationStatus` | Specifies the type of validation error for a proposed mention, if any.<br>• `Disallowed`—The proposed mention is invalid and will be 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.<br>• `Inaccessible`—The proposed mention is allowed but the user or record being mentioned will not be notified because they don't have access to the parent record being discussed.<br>• `Ok`—There is no validation error for this proposed mention. |
| `ConnectApi.MessageSegmentType` | Specifies the type of message segment, such as text, link, field change name, or field change value.<br>• `EntityLink`<br>• `FieldChange`<br>• `FieldChangeName`<br>• `FieldChangeValue`<br>• `Hashtag`<br>• `Link`<br>• `Mention`<br>• `MoreChanges`<br>• `ResourceLink`<br>• `Text` |
| `ConnectApi.RecordColumnOrder` | The order in which fields are rendered in a grid.<br>• `LeftRight`—Fields are rendered from left to right.<br>• `TopDown`—Fields are rendered from the top down. |
| `ConnectApi.RecordFieldType` | The data type of a record field.<br>• `Address`<br>• `Blank`<br>• `Boolean`<br>• `Compound`<br>• `CreatedBy`<br>• `Date`<br>• `DateTime`<br>• `Email` |

| Enum | Description |
| --- | --- |
| | • LastModifiedBy<br>• Location<br>• Name<br>• Number<br>• Percent<br>• Phone<br>• Picklist<br>• Reference<br>• Text<br>• Time |
| ConnectApi.SortOrder | A generic sort order direction.<br>• Ascending—Ascending order (A-Z).<br>• Descending—Descending order (Z-A). |
| ConnectApi.TopicSort | Specifies the order returned by the sort:<br>• popularDesc—Sorts topics by popularity with the most popular first. This value is the default.<br>• alphaAsc—Sorts topics alphabetically. |
| ConnectApi.UserProfileTabType | Specifies the type of user profile tab:<br>• CustomVisualForce—Tab that displays data from a Visualforce page.<br>• CustomWeb—Tab that displays data from any external Web-based application or Web page.<br>• Feed—Tab that displays the Chatter feed.<br>• Overview—Tab that displays user details. |
| ConnectApi.UserType | Specifies the type of user.<br>• ChatterGuest—User is a Chatter customer in an external group<br>• ChatterOnly—User is a Chatter Free customer<br>• Guest—Unauthenticated users<br>• Internal—User is a standard organization member<br>• Portal—User is a Customer Portal User, a communities user, and so on.<br>• System—User is Chatter Expert or a system user<br>• Undefined—User is a user type that is a custom object. |
| ConnectApi.WorkflowProcessStatus | Specifies the status of a workflow process.<br>• Approved<br>• Fault<br>• Held<br>• NoResponse<br>• Pending<br>• Reassigned<br>• Rejected<br>• Removed |

| Enum | Description |
|---|---|
| | • `Started` |
| `ConnectApi.ZoneSearchResultType` | Specifies the zone search result type.<br>• `Article`—Search results contain only articles.<br>• `Question`—Search results contain only questions. |
| `ConnectApi.ZoneShowIn` | Specifies the zone search result type.<br>• `Community`—Available in a community.<br>• `Internal`—Available internally only.<br>• `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 779.

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.

### *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.

***EmptyRecycleBinResult Class***

The result of the emptyRecycleBin DML operation returned by the `Database.emptyRecycleBin` method.

***Error Class***

Contains 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.

***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`.

***execute(Database.BatchableContext, List<sObject>)***

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(Database.BatchableContext)***

Gets invoked when the batch job finishes. Place any clean up code in this method.

***start(Database.BatchableContext)***

Gets invoked when the batch job starts. Returns the record set as an iterable that will be batched for execution.

***start(Database.BatchableContext)***

Gets invoked when the batch job starts. Returns the record set as a QueryLocator object that will be batched for execution.

## execute(Database.BatchableContext, List<sObject>)

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 context, List<sObject> scope)
```

### Parameters

***context***

Type: Database.BatchableContext

Contains the job ID.

***scope***

Type: List<sObject>

Contains the batch of records to process.

### Return Value

Type: Void

## finish(Database.BatchableContext)

Gets invoked when the batch job finishes. Place any clean up code in this method.

### Signature

```
public Void finish(Database.BatchableContext context)
```

### Parameters

***context***

Type: Database.BatchableContext

Contains the job ID.

### Return Value

Type: Void

### start(Database.BatchableContext)

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 context)
```

#### Parameters

***context***

Type: Database.BatchableContext

Contains the job ID.

#### Return Value

Type: System.Iterable

### start(Database.BatchableContext)

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 context)
```

#### Parameters

***context***

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`.

***getChildJobId()***

Returns the ID of the current batch job chunk that is being processed.

***getJobId()***

Returns the batch job ID.

### 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.

***getDeletedDate()***

Returns the deleted date of this record.

***getId()***

Returns the ID of a record deleted within the time window specified in the `Database.getDeleted` method.

### 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.

## DeleteResult Methods

The following are methods for `DeleteResult`. All are instance methods.

#### *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 Database.com user interface.

## DmlOptions Properties

The following are properties for `DmlOptions`.

***allowFieldTruncation***

Specifies the truncation behavior of large strings.

***localeOptions***

Specifies the language of any labels that are returned by Apex.

***optAllOrNone***

Specifies whether the operation allows for partial success.

### 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.

### 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 Database.com supports, see What languages does Database.com support? in the Database.com 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.com API version 20.0 and later.

## 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.

***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.

***getId()***

Returns the ID of the record or sObject you attempted to delete.

***isSuccess()***

Returns `true` if the record or sObject was successfully removed from the Recycle Bin; otherwise `false`.

### 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

Contains information about an error that occurred during a DML operation when using a Database method.

### Namespace

Database

## Error Methods

The following are methods for `Error`. All are instance methods.

***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.

### 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 Database.com WSDLs and Client Authentication Certificates in the Database.com 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.

***getDeletedRecords()***

Returns a list of deleted records for the time window specified in the `Database.getDeleted` method call.

***getEarliestDateAvailable()***

Returns the date in Coordinated Universal Time (UTC) of the earliest physically deleted object for the sObject type specified in `Database.getDeleted`.

***getLatestDateCovered()***

Returns the date in Coordinated Universal Time (UTC) of the last date covered in the `Database.getDeleted` call.

## 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.

***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

# 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.

***getQuery()***

Returns the query used to instantiate the `Database.QueryLocator` object. This is useful when testing the `start` method.

***iterator()***

Returns a new instance of a query locator iterator.

### 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 Invoice_Statement__c]).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 invoice statements. 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 Invoice_Statement__c LIMIT 5]);
// Get an iterator
Database.QueryLocatorIterator it =  q.iterator();

// Iterate over the records
while (it.hasNext())
{
    Invoice_Statement__c a = (Invoice_Statement__c)it.next();
    System.debug(a);
}
```

# QueryLocatorIterator Methods

The following are methods for `QueryLocatorIterator`. All are instance methods.

***hasNext()***

Returns `true` if there are one or more records remaining in the collection; otherwise, returns `false`.

***next()***

Advances the iterator to the next sObject record and returns the sObject.

## 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:

```
Invoice_Statement__c a =
  (Invoice_Statement__c)myIterator.next();
```

### Example

```
Invoice_Statement__c a =
  (Invoice_Statement__c)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.

# SaveResult Methods

The following are methods for `SaveResult`. All are instance methods.

***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

## 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.

***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 undelete.

***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.

***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 update or insert.

***isCreated()***

A Boolean value that is set to `true` if the record was created, `false` if the record was updated.

***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 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

## Dom Namespace

The `Dom` namespace provides classes and methods for approval processing.

The following are the classes in the `Dom` namespace.

***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.

***Document Constructors***

***Document Methods***

## Document Constructors

The following are constructors for `Document`.

***Document()***

Creates a new instance of the `Dom.Document` class.

### 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.

***createRootElement(String, String, String)***

Creates the top-level root element for a document.

***getRootElement()***

Returns the top-level root element node in the document. If this method returns `null`, the root element has not been created yet.

***load(String)***

Parse the XML representation of the document specified in the *`xml`* argument and load it into a document.

***toXmlString()***

Returns the XML representation of the document as a String.

### createRootElement(String, String, String)

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(String)

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.

***addChildElement(String, String, String)***

Creates a child element node for this node.

***addCommentNode(String)***

Creates a child comment node for this node.

***addTextNode(String)***

Creates a child text node for this node.

***getAttribute(String, String)***

Returns *`namespacePrefix:attributeValue`* for the given key and key namespace.

***getAttributeCount()***

Returns the number of attributes for this node.

***getAttributeKeyAt(Integer)***

Returns the attribute key for the given index. Index values start at 0.

***getAttributeKeyNsAt(Integer)***

Returns the attribute key namespace for the given index.

***getAttributeValue(String, String)***

Returns the attribute value for the given key and key namespace.

***getAttributeValueNs(String, String)***

Returns the attribute value namespace for the given key and key namespace.

***getChildElement(String, String)***

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(String)***

Returns the namespace of the element for the given prefix.

***getNodeType()***

Returns the node type.

***getParent()***

Returns the parent of this element.

***getPrefixFor(String)***

Returns the prefix of the given namespace.

***getText()***

Returns the text for this node.

***removeAttribute(String, String)***

Removes the attribute with the given key and key namespace. Returns `true` if successful, `false` otherwise.

***removeChild(Dom.XmlNode)***

Removes the given child node.

***setAttribute(String, String)***

Sets the key attribute value.

***setAttributeNs(String, String, String, String)***

Sets the key attribute value.

***setNamespace(String, String)***

Sets the namespace for the given prefix.

## addChildElement(String, String, String)

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, Database.com 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(String)

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(String)

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(String, String)

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(Integer)

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(Integer)

Returns the attribute key namespace for the given index.

#### Signature

```
public String getAttributeKeyNsAt(Integer index)
```

#### Parameters

***index***

Type: Integer

#### Return Value

Type: String

### getAttributeValue(String, String)

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(String, String)

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(String, String)

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(String)

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(String)

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(String, String)

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(Dom.XmlNode)

Removes the given child node.

#### Signature

```
public Boolean removeChild(Dom.XmlNode childNode)
```

#### Parameters

***childNode***

Type: Dom.XmlNode

#### Return Value

Type: Boolean

### setAttribute(String, String)

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(String, String, String, String)

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(String, String)

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

# QuickAction Namespace

The `QuickAction` namespace provides classes and methods for publisher actions.

The following are the classes in the `QuickAction` namespace.

***DescribeAvailableQuickActionResult Class***

Contains describe metadata information for a publisher 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 publisher quick action.

***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 publisher action with the `QuickAction` class, use the `QuickActionResult` class for processing action results.

# DescribeAvailableQuickActionResult Class

Contains describe metadata information for a publisher 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`).

**Note:** In the application, QuickActions are referred to as actions or publisher actions.

# DescribeAvailableQuickActionResult Methods

The following are methods for `DescribeAvailableQuickActionResult`. All are instance methods.

***getLabel()***

The publisher action label.

***getName()***

The publisher action name.

***getType()***

The publisher action type.

## getLabel()

The publisher action label.

### Signature

```
public String getLabel()
```

### Return Value

Type: String

## getName()

The publisher action name.

### Signature

```
public String getName()
```

#### Return Value

Type: String

### getType()

The publisher 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

### Usage

**Note:** In the application, QuickActions are referred to as actions or publisher actions.

## DescribeLayoutComponent Methods

The following are methods for `DescribeLayoutComponent`. All are instance methods.

***getDisplayLines()***

Returns the vertical lines displayed for a field. Applies to `textarea` and multi-select picklist fields.

***getTabOrder()***

Returns the tab order for the item in the row.

***getType()***

Returns the name of the `QuickAction.DescribeLayoutComponent` type for this component.

***getValue()***

Returns the name of the field if the type for `QuickAction.DescribeLayoutComponent` is `textarea`.

### 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`.

**Note:** In the application, QuickActions are referred to as actions or publisher actions.

## DescribeLayoutItem Methods

The following are methods for `DescribeLayoutItem`. All are instance methods.

***getLabel()***

Returns the label text for this item.

***getLayoutComponents()***

Returns a list of `QuickAction.DescribeLayoutComponents` for this item.

***isEditable()***

Indicates whether this item can be edited (`true`) or not (`false`).

***isPlaceholder()***

Indicates whether this item is a placeholder (`true`) or not (`false`). If `true`, then this item is blank.

***isRequired()***

Indicates whether this item is required (`true`) or not (`false`).

### 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).

**Note:** In the application, QuickActions are referred to as actions or publisher actions.

## DescribeLayoutRow Methods

The following are methods for `DescribeLayoutRow`. All are instance methods.

***getLayoutItems()***

Returns either a specific field or an empty `QuickAction.DescribeLayoutItem` (one that contains no `QuickAction.DescribeLayoutComponent` objects).

***getNumItems()***

Returns the number of `QuickAction.DescribeLayoutItem`.

### 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`.

***getColumns()***

Returns the number of columns in the `QuickAction.DescribeLayoutSection`.

***getHeading()***

The heading text (label) for the `QuickAction.DescribeLayoutSection`.

***getLayoutRows()***

Returns an array of one or more `QuickAction.DescribeLayoutRow` objects.

***getRows()***

Returns the number of rows in the `QuickAction.DescribeLayoutSection`.

***isUseCollapsibleSection()***

Indicates whether the `QuickAction.DescribeLayoutSection` is a collapsible section (`true`) or not (`false`).

***isUseHeading()***

Indicates whether to use the `heading` (`true`) or not (`false`).

## 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.

**Note:** In the application, QuickActions are referred to as actions or publisher actions.

## DescribeQuickActionDefaultValue Methods

The following are methods for `DescribeQuickActionDefaultValue`. All are instance methods.

***getDefaultValue()***

Returns the default value of the quick action.

***getField()***

Returns the field name of the action.

### 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 publisher quick action.

### Namespace

QuickAction

### Usage

The QuickAction `describeQuickActions` method returns an array of quick action describe result objects (`QuickAction.DescribeQuickActionResult`).

**Note:** In the application, QuickActions are referred to as actions or publisher actions.

## DescribeQuickActionResult Methods

The following are methods for `DescribeQuickActionResult`. All are instance methods.

***getCanvasApplicationName()***

Returns the name of the Canvas application, if used.

***getDefaultValues()***

Returns the default values for a action.

***getHeight()***

Returns the height in pixels of the action pane.

***getIconName()***

Returns the actions' icon name.

***getIconUrl()***

Returns the URL of the 32x32 icon used for the action.

***getIcons()***

Returns a list of `Schema.DescribeIconResult` objects that describe colors used in a tab.

***getLabel()***

Returns the action label.

***getLayout()***

Returns the layout sections that comprise an action.

***getMiniIconUrl()***

Returns the 16x16 icon URL.

***getName()***

Returns the action name.

***getSourceSobjectType()***

Returns the object type used for the action.

***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.

***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

### getWidth()

If a custom action is created, returns the width in pixels of the action pane.

#### Signature

```
public Integer getWidth()
```

#### Return Value

Type: Integer

## 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.com 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();
```

**Note:** In the application, QuickActions are referred to as actions or publisher actions.

### 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);
```

***QuickActionRequest Constructors***
***QuickActionRequest Methods***

**See Also:**

*QuickAction Class*

## QuickActionRequest Constructors

The following are constructors for `QuickActionRequest`.

***QuickActionRequest()***

Creates a new instance of the `QuickAction.QuickActionRequest` class.

### 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.

***getContextId()***

Returns this QuickAction's context record ID.

***getQuickActionName()***

Returns this QuickAction's name.

***getRecord()***

Returns the QuickAction's associated record.

***setContextId(Id)***

Sets this QuickAction's context ID. Returned by `getContextId`.

***setQuickActionName(String)***

Sets this QuickAction's name. Returned by `getQuickActionName`.

***setRecord(SObject)***

Sets a record for this QuickAction. Returned by `getRecord`.

### 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(Id)

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 Salesforce.comAPI version 28.0, sets this QuickAction's parent ID and is returned by `getParentId`.

### setQuickActionName(String)

Sets this QuickAction's name. Returned by `getQuickActionName`.

#### Signature

```
public Void setQuickActionName(String name)
```

#### Parameters

***name***

Type: String

#### Return Value

Type: Void

### setRecord(SObject)

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 publisher action with the `QuickAction` class, use the `QuickActionResult` class for processing action results.

## Namespace

QuickAction

## Usage

**Note:** In the application, QuickActions are referred to as actions or publisher actions.

**See Also:**

*QuickAction Class*

# QuickActionResult Methods

The following are methods for `QuickActionResult`. All are instance methods.

***getErrors()***

If an error occurs, an array of one or more database error objects, along with error codes and descriptions, is returned.

***getIds()***

The IDs of the QuickActions being processed.

***isCreated()***

Returns `true` if the action is created; otherwise, `false`.

***isSuccess()***

Returns `true` if the action completes successfully; otherwise, `false`.

## 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

## Schema Namespace

The `Schema` namespace provides classes and methods for schema metadata information.

The following are the classes in the `Schema` namespace.

***ChildRelationship Class***

Contains methods for accessing the child relationship as well as the child sObject for a parent sObject.

***DescribeFieldResult Class***

Contains methods for describing sObject fields.

***DescribeSObjectResult Class***

Contains methods for describing sObjects.

***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.

#### 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

### Usage

You can only use 100 `getChildRelationships` method calls per Apex request. For more information about governor limits, see Understanding Execution Governors and Limits on page 203.

### Example

A ChildRelationship object is returned from the sObject describe result using the `getChildRelationship` method. For example:

```
Schema.DescribeSObjectResult R = Invoice_Statement__c.SObjectType.getDescribe();
List<Schema.ChildRelationship> C = R.getChildRelationships();
```

## ChildRelationship Methods

The following are methods for `ChildRelationship`. All are instance methods.

#### getChildSObject()

Returns the token of the child sObject on which there is a foreign key back to the parent sObject.

#### getField()

Returns the token of the field that has a foreign key back to the parent sObject.

#### getRelationshipName()

Returns the name of the relationship.

#### isCascadeDelete()

Returns `true` if the child object is deleted when the parent object is deleted, `false` otherwise.

***isDeprecatedAndHidden()***

Reserved for future use.

***isRestrictedDelete()***

Returns `true` if the parent object can't be deleted because it is referenced by a child object, `false` otherwise.

## 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

## 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 F = Invoice_Statement__c.Description__c.getDescribe();
```

## DescribeFieldResult Methods

The following are methods for `DescribeFieldResult`. All are instance methods.

***getByteLength()***

For variable-length fields (including binary fields), returns the maximum size of the field, in bytes.

***getCalculatedFormula()***

Returns the formula specified for this field.

***getController()***

Returns the token of the controlling field.

***getDefaultValue()***

Returns the default value for this field.

***getDefaultValueFormula()***

Returns the default value specified for this field if a formula is not used.

### getDigits()

Returns the maximum number of digits specified for the field. This method is only valid with Integer fields.

### getInlineHelpText()

Returns the content of the field-level help.

### getLabel()

Returns the text label of the field. This label can be localized.

### getLength()

For string fields, returns the maximum size of the field in Unicode characters (not bytes).

### 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.

### getName()

Returns the field name used in Apex.

### getPicklistValues()

Returns a list of PicklistEntry objects. A runtime error is returned if the field is not a picklist.

### 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).

### 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.

### getRelationshipName()

Returns the name of the relationship.

### 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. 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 Database.com online help.

## getLabel()

Returns the text label of the field. This label can be localized.

### Signature

```
public String getLabel()
```

### Return Value

Type: String

### Usage

## 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

### 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 *Database.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 *Database.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 Database.com 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`, Database.com 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 custom objects. Objects can only have one name field.

## isNamePointing()

Returns `true` if the field can have multiple types of objects as parents. 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

## 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.

### 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.

### 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.

### 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.

### 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.

### 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 Salesforce.comAPI version 19.0 and later.

### 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

For more information, see Understanding Apex Describe Information.

### Example

```
Schema.DescribeFieldResult F =
Schema.SObjectType.Merchandise__c.fields.Name;
```

## 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 Invoice_Statement__c object has child relationship `Line_Items__r`.

## 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.

### 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 Database.com user interface. See the Database.com 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 Database.com user interface. See the Database.com 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

## 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

## 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 Salesforce.comAPI version 19.0 and later.

#### Signature

```
public Boolean isFeedEnabled()
```

#### 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

## 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) |
| `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 |

| Type Field Value | What the Field Object Contains |
|---|---|
| `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 Database.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.Merchandise__c.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 =
  Merchandise__c.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.Merchandise__c.fieldSets.getMap().get('field_set_name');
Schema.FieldSet fs2 = Schema.SObjectType.Merchandise__c.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.

***getDescription()***

Returns the field set's description.

***getFields()***

Returns a list of `Schema.FieldSetMember` objects for the fields making up the field set.

***getLabel()***

Returns the text label of the field.

***getName()***

Returns the field set's name.

***getNamespace()***

Returns the field set's namespace.

***getSObjectType()***

Returns the `Schema.sObjectType` of the sObject containing the field set definition.

### 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 of the field.

#### 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.Merchandise__c.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.Merchandise__c.fieldSets.field_set_name.getFields();
```

**See Also:**

*FieldSet Class*

## FieldSetMember Methods

The following are methods for `FieldSetMember`. All are instance methods.

***getDBRequired()***

Returns `true` if the field is required by the field's definition in its sObject, otherwise, `false`.

***getFieldPath()***

Returns a field path string in a format ready to be used in a dynamic SOQL query.

***getLabel()***

Returns the text label of the field.

***getRequired()***

Returns `true` if the field is required by the field set, otherwise, `false`.

***getType()***

Returns the field's Apex data type.

### 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 of the field.

#### 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. 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 = Invoice_Statement__c.Status__c.getDescribe();
List<Schema.PicklistEntry> P = F.getPicklistValues();
```

You can only use 100 `getPicklistValue` method calls per Apex request. For more information about governor limits, see Understanding Execution Governors and Limits on page 203.

## PicklistEntry Methods

The following are methods for `PicklistEntry`. All are instance methods.

***getLabel()***

Returns the display name of this item in the picklist.

***getValue()***

Returns the value of this item in the picklist.

***isActive()***

Returns `true` if this item must be displayed in the drop-down list for the picklist field in the user interface, `false` otherwise.

***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.

### 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

## 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 = Invoice_Statement__c.Status__c.getDescribe();
Schema.sObjectField T = F.getSObjectField();
```

## sObjectField Methods

The following are instance methods for `sObjectField`.

***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 = Invoice_Statement__c.Status__c.getDescribe();
List<Schema.sObjectType> P = F.getReferenceTo();
```

## SObjectType Methods

The following are methods for `SObjectType`. All are instance methods.

***getDescribe()***

Returns the describe sObject result for this field.

***newSObject()***

Constructs a new sObject of this type.

***newSObject(ID)***

Constructs a new sObject of this type, with the specified ID.

***newSObject(ID, Boolean)***

Constructs a new sObject of this type, and optionally, of the specified record type ID and with default custom field values.

### 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(ID, Boolean)

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 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 Database.com online help.

#### Example: Creating New sObject with Default Values

This sample creates a merchandise item with default values using the `newSObject` method. It also creates a second merchandise item for a specific record type. For both merchandise items, the sample sets the Name field, which is a required field that doesn't have a default value, before inserting the new items. The sample assumes that all other required fields have default values predefined.

```
// Create an account with predefined default values
Merchandise__c m = (Merchandise__c)Merchandise__c.sObjectType.newSObject(null, true);
// Provide a value for Name
m.Name = 'Erasers';
// Insert new merchandise
insert m;

// This is for record type RT1 of Merchandise__c
ID rtId = [SELECT Id FROM RecordType WHERE sObjectType='Merchandise__c' AND Name='RT1'].Id;
// Create an account with predefined default values
Merchandise__c m2 = (Merchandise__c)Merchandise__c.sObjectType.newSObject(rtId, true);
// Provide a value for Name
m2.Name = 'Rulers';
// Insert new merchandise
insert m2;
```

# System Namespace

The `System` namespace provides classes and methods for core Apex functionality.

The following are the classes in the `System` namespace.

### *Blob Class*

Contains methods for the Blob primitive data type.

### *Boolean Class*

Contains methods for the Boolean primitive data type.

### *Comparable Interface*

Adds sorting support for Lists that contain non-primitive types, that is, Lists of user-defined types.

### *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, 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 Methods*

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.

### 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.

### 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.

### Pattern Class

Represents a compiled representation of a regular expression.

### QuickAction Class

Use Apex to request and process publisher actions on objects that allow custom fields, on objects that appear in a Chatter feed, or on objects that are available globally.

### 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

Used with dynamic SOSL queries.

### Set Class

Represents a collection of unique elements with no duplicate values.

### sObject Class

Contains methods for the sObject data type.

### *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.

### *Type Class*

Contains methods for getting the Apex type that corresponds to an Apex class and for instantiating new types.

### *URL Class*

Represents a uniform resource locator (URL) and provides access to parts of the URL. Enables access to the base URL of a Database.com organization.

### *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.

### *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.

# Blob Class

Contains methods for the Blob primitive data type.

## Namespace

System

## Usage

For more information on Blobs, see Primitive Data Types on page 22.

# Blob Methods

The following are methods for `Blob`.

***size()***

Returns the number of characters in the Blob.

***toPdf(String)***

Creates a binary object out of the given string, encoding it as a PDF file.

***toString()***

Casts the Blob into a String.

***valueOf(String)***

Casts the specified String to a Blob.

## 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(String)

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

### toString()

Casts the Blob into a String.

#### Signature

```
public String toString()
```

#### Return Value

Type: String

### valueOf(String)

Casts the specified String to a Blob.

#### Signature

```
public static Blob valueOf(String toBlob)
```

#### Parameters

***toBlob***

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.

***valueOf(String)***

Converts the specified string to a Boolean value and returns `true` if the specified string value is `true`. Otherwise, returns `false`.

***valueOf(Object)***

Converts the specified history tracking field value to a Boolean value.

### valueOf(String)

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 toBoolean)
```

#### Parameters

***toBoolean***

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(Object)

Converts the specified history tracking field value to 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 when the field type corresponds to a Boolean type, like a checkbox field.

## 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`.

***Comparable Methods***
***Comparable Example Implementation***

**See Also:**

*List Class*

## Comparable Methods

The following are methods for `Comparable`.

***compareTo(Object)***

Returns an Integer value that is the result of the comparison.

### compareTo(Object)

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);
    }
}
```

# 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.

#### *decrypt(String, Blob, Blob, Blob)*

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.

#### *decryptWithManagedIV(String, Blob, Blob)*

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.

#### *encrypt(String, Blob, Blob, Blob)*

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.

#### *encryptWithManagedIV(String, Blob, Blob)*

Encrypts the Blob `clearText` using the specified algorithm and private key. Use this method when you want Database.com to generate the initialization vector for you.

#### *generateAesKey(Integer)*

Generates an Advanced Encryption Standard (AES) key.

#### *generateDigest(String, Blob)*

Computes a secure, one-way hash digest based on the supplied input string and algorithm name.

#### *generateMac(String, Blob, Blob)*

Computes a message authentication code (MAC) for the input string, using the private key and the specified algorithm.

#### *getRandomInteger()*

Returns a random Integer.

#### *getRandomLong()*

Returns a random Long.

#### *sign(String, Blob, Blob)*

Computes a unique digital signature for the input string, using the supplied private key and the specified algorithm.

## decrypt(String, Blob, Blob, Blob)

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.)

### decryptWithManagedIV(String, Blob, Blob)

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.

## encrypt(String, Blob, Blob, Blob)

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 Database.com 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.

## encryptWithManagedIV(String, Blob, Blob)

Encrypts the Blob *clearText* using the specified algorithm and private key. Use this method when you want Database.com 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.

### generateAesKey(Integer)

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

## generateDigest(String, Blob)

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

## generateMac(String, Blob, Blob)

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

### getRandomInteger()

Returns a random Integer.

#### Signature

```
public static Integer getRandomInteger()
```

#### Return Value

Type: Integer

### getRandomLong()

Returns a random Long.

#### Signature

```
public static Long getRandomLong()
```

#### Return Value

Type: Long

### sign(String, Blob, Blob)

Computes a unique digital signature for the input string, using the supplied private key and the specified algorithm.

#### Signature

```
public static Blob sign(String algorithmName, Blob input, Blob privateKey)
```

#### Parameters

***algorithmName***

Type: String

The valid values for *algorithmName* are `RSA-SHA1` or `RSA`. Both values represent the same algorithm.

***input***

Type: Blob

***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 is an example declaration and initialization.

```
String algorithmName = 'RSA';
String key = '';
Type: Blob privateKey = EncodingUtil.base64Decode(key);
Type: Blob input = Type: Blob.valueOf('12345qwerty');
Crypto.sign(algorithmName, input, privateKey);
```

# 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, 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 Database.com user interface, see “Custom Settings Overview” in the Database.com 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 list custom setting, Foundation_Countries, that has a single field, `Country_Code`. 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);
```

***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.

***getAll()***

Returns a map of the data sets defined for the custom setting.

***getInstance(String)***

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getValues(`***`dataset_name`***`)`.

***getValues(String)***

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getInstance(`***`dataset_name`***`)`.

### 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 Salesforce.comAPI 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 Salesforce.comAPI 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(String)

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getValues(dataset_name)`.

#### Signature

```
public CustomSetting__c getInstance(String dataset_name)
```

#### Parameters

***dataset_name***

Type: String

#### Return Value

Type: CustomSetting__c

#### Usage

If no data is defined for the specified data set, this method returns `null`.

### getValues(String)

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getInstance(dataset_name)`.

#### Signature

```
public CustomSetting__c getValues(String dataset_name)
```

#### Parameters

***dataset_name***

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.

***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.

***getInstance(ID)***

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.

***getInstance(ID)***

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.

***getOrgDefaults()***

Returns the custom setting data set record for the organization.

***getValues(ID)***

Returns the custom setting data set record for the specified user ID.

***getValues(ID)***

Returns the custom setting data set for the specified profile ID.

## 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.com 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 Administrator 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(ID)

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 user_Id)
```

### Parameters

***user_Id***

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.com 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(ID)

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 profile_Id)
```

### Parameters

***profile_Id***

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 Salesforce.comAPI 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 Salesforce.comAPI version 21.0 or earlier, this method returns `null` if no custom setting data is defined for the organization.

### getValues(ID)

Returns the custom setting data set record for the specified user ID.

#### Signature

```
public CustomSetting__c getValues(ID user_Id)
```

#### Parameters

***user_Id***
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(`***`User_Id`***`)` returns `null` for this custom setting field.

### getValues(ID)

Returns the custom setting data set for the specified profile ID.

#### Signature

```
public CustomSetting__c getValues(ID Profile_Id)
```

#### Parameters

***profile_Id***

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(Profile_Id)` 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.

***countQuery(String)***

Returns the number of records that a dynamic SOQL query would return when executed.

***delete(SObject, Boolean)***

Deletes an existing sObject record from your organization's data.

***delete(SObject[], Boolean)***

Deletes a list of existing sObject records from your organization’s data.

***delete(ID, Boolean)***

Deletes existing sObject records from your organization’s data.

***delete(ID[], Boolean)***

Deletes a list of existing sObject records from your organization’s data.

***emptyRecycleBin(ID[])***

Permanently deletes the specified records from the Recycle Bin.

### *emptyRecycleBin(sObject)*

Permanently deletes the specified sObject from the Recycle Bin.

### *emptyRecycleBin(sObject[])*

Permanently deletes the specified sObjects from the Recycle Bin.

### *executeBatch(sObject)*

Executes the specified class as a batch Apex job.

### *executeBatch(sObject, Integer)*

Executes the specified class as a batch Apex job.

### *getDeleted(String, Datetime, Datetime)*

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.

### *getQueryLocator(sObject [])*

Creates a QueryLocator object used in batch Apex.

### *getQueryLocator(String)*

Creates a QueryLocator object used in batch Apex.

### *getUpdated(String, Datetime, Datetime)*

Returns the list of individual records that have been updated for an sObject type within the specified start and end dates and times.

### *insert(sObject, Boolean)*

Adds an sObject to your organization's data.

### *insert(sObject [], Boolean)*

Adds one or more sObjects to your organization’s data.

### *insert(sObject, Database.DMLOptions)*

Adds an sObject to your organization's data.

### *insert(sObject[], Database.DMLOptions)*

Adds an sObject to your organization's data.

### *query(String)*

Creates a dynamic SOQL query at runtime.

### *rollback(System.Savepoint)*

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(sObject, Boolean)*

Restores an existing sObject record from your organization's Recycle Bin.

***undelete(sObject [], Boolean)***

Restores one or more existing sObject records, such as individual invoice statements.

***undelete(ID, Boolean)***

Restores an existing sObject record from your organization's Recycle Bin.

***undelete(ID[], Boolean)***

Restores one or more existing sObject records, such as individual invoice statements.

***update(sObject, Boolean)***

Modifies an existing sObject record in your organization's data.

***update(sObject [], Boolean)***

Modifies one or more existing sObject records, such as individual invoice statements, in your organization's data.

***update(sObject, Database.DmlOptions)***

Modifies an existing sObject record in your organization's data.

***update(sObject [], Database.DMLOptions)***

Modifies one or more existing sObject records, such as individual invoice statements, in your organization's data.

***upsert(sObject, Schema.SObjectField, Boolean)***

Creates a new sObject record or updates an existing sObject record within a single statement, using an optional custom field to determine the presence of existing objects.

***upsert(sObject[], Schema.SObjectField, Boolean)***

Creates new sObject records or updates existing sObject records within a single statement, using an optional custom field to determine the presence of existing objects.

## countQuery(String)

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 135.

Each executed `countQuery` method counts against the governor limit for SOQL queries.

### Example

```
String QueryString =
    'SELECT count() ' +
    'FROM Invoice_Statement__c';
Integer i =
    Database.countQuery(QueryString);
```

## delete(SObject, Boolean)

Deletes an existing sObject record from your organization's data.

### Signature

```
public static Database.DeleteResult delete(SObject recordToDelete, Boolean opt_allOrNone)
```

### Parameters

***recordToDelete***

Type: sObject

***opt_allOrNone***

Type: Boolean

The optional *opt_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(SObject[], Boolean)

Deletes a list of existing sObject records from your organization's data.

### Signature

```
public static Database.DeleteResult[] delete(SObject[] recordsToDelete, Boolean opt_allOrNone)
```

### Parameters

***recordsToDelete***

Type: sObject[]

***opt_allOrNone***

Type: Boolean

The optional *opt_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 all merchandise items named 'Pencil':

```
Merchandise__c[] pencils =
  [SELECT Id, Name
   FROM Merchandise__c
   WHERE Name = 'Pencil'];
Database.DeleteResult[] DR_Dels =
  Database.delete(pencils);
```

### delete(ID, Boolean)

Deletes existing sObject records from your organization's data.

#### Signature

```
public static Database.DeleteResult delete(ID recordID, Boolean opt_allOrNone)
```

#### Parameters

***recordID***

Type: ID

***opt_allOrNone***

Type: Boolean

The optional *opt_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(ID[], Boolean)

Deletes a list of existing sObject records from your organization's data.

#### Signature

```
public static Database.DeleteResult[] delete(ID[] recordIDs, Boolean opt_allOrNone)
```

#### Parameters

***recordIDs***

Type: ID[]

***opt_allOrNone***

Type: Boolean

The optional *opt_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(ID[])

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.
- 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(sObject)

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.
- 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(sObject[])

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.

- 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(sObject)

Executes the specified class as a batch Apex job.

### Signature

```
public static ID executeBatch(sObject className)
```

### Parameters

***className***

Type: sObject

### Return Value

Type: ID

### Usage

For more information, see Using Batch Apex on page 179.

**Note:** The class called by the `executeBatch` method implements the `execute` method.

When calling this method, Database.com 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`.

## executeBatch(sObject, Integer)

Executes the specified class as a batch Apex job.

### Signature

```
public static ID executeBatch(sObject className, Integer scope)
```

### Parameters

***className***

Type: sObject

***scope***

Type: Integer

### Return Value

Type: ID

### Usage

The value for *scope* must be greater than 0.

**Note:** The class called by the `executeBatch` method implements the `execute` method.

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, Database.com 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 179.

## getDeleted(String, Datetime, Datetime)

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 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(sObject [])

Creates a QueryLocator object used in batch Apex.

### 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.

## getQueryLocator(String)

Creates a QueryLocator object used in batch Apex.

### 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.

## getUpdated(String, Datetime, Datetime)

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 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(sObject, Boolean)

Adds an sObject to your organization's data.

### Signature

```
public static Database.SaveResult insert(sObject recordToInsert, Boolean opt_allOrNone)
```

### Parameters

***recordToInsert***

Type: sObject

***opt_allOrNone***

Type: Boolean

The optional *opt_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(sObject [], Boolean)

Adds one or more sObjects to your organization's data.

#### Signature

```
public static Database.SaveResult[] insert(sObject [] recordsToInsert, Boolean opt_allOrNone)
```

#### Parameters

***recordsToInsert***

Type: sObject []

***opt_allOrNone***

Type: Boolean

The optional *opt_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.

The optional *opt_DMLOptions* parameter specifies additional data for the transaction, such as rollback behavior when errors occur during record insertions.

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 invoice statements:

```
Invoice_Statement__c inv1 =
  new Invoice_Statement__c(
    Description__c = 'Invoice 1');
Database.SaveResult[] lsr =
  Database.insert(
    new Invoice_Statement__c[]{
      inv1,
```

```
      new Invoice_Statement__c(
        Description__c = 'Invoice 2')},
    false);
```

## insert(sObject, Database.DMLOptions)

Adds an sObject to your organization's data.

### Signature

```
public static Database.SaveResult insert(sObject recordToInsert, Database.DMLOptions options)
```

### Parameters

***recordToInsert***

Type: sObject

***options***

Type: Database.DMLOptions

The optional *opt_DMLOptions* parameter specifies additional data for the transaction, such as 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(sObject[], Database.DMLOptions)

Adds an sObject to your organization's data.

### Signature

```
public static Database.SaveResult insert(sObject[] recordToInsert, Database.DMLOptions
options)
```

### Parameters

***recordToInsert***

Type: sObject

***options***

Type: Database.DMLOptions

The optional *opt_DMLOptions* parameter specifies additional data for the transaction, such as 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.

### query(String)

Creates a dynamic SOQL query at runtime.

#### Signature

```
public static sObject[] query(String query)
```

#### Parameters

***query***

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 135.

Each executed `query` method counts against the governor limit for SOQL queries.

### rollback(System.Savepoint)

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 sp)
```

#### Parameters

***sp***

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 execution context. If you declare a savepoint as a static variable then try to use it across trigger contexts you will receive a runtime error.
- Each savepoint you set counts against the governor limit for DML statements.

For an example, see Transaction Control.

### undelete(sObject, Boolean)

Restores an existing sObject record from your organization's Recycle Bin.

#### Signature

```
public static Database.UndeleteResult undelete(sObject recordToUndelete, Boolean
opt_allOrNone)
```

#### Parameters

***recordToUndelete***

Type: sObject

***opt_allOrNone***

Type: Boolean

The optional *opt_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(sObject [], Boolean)

Restores one or more existing sObject records, such as individual invoice statements.

#### Signature

```
public static Database.UndeleteResult[] undelete(sObject [] recordsToUndelete, Boolean
opt_allOrNone)
```

#### Parameters

***recordsToUndelete***

Type: sObject []

***opt_allOrNone***

Type: Boolean

The optional *opt_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 invoice statements with the specified description. The `ALL ROWS` keyword queries all rows for both top-level and aggregate relationships, including deleted records and archived activities.

```
Invoice_Statement__c[] SavedInvoices =
  [SELECT Id
   FROM Invoice_Statement__c
   WHERE Description__c = 'My invoice'
   ALL ROWS];
Database.UndeleteResult[] UDR_Dels =
  Database.undelete(SavedInvoices);
```

## undelete(ID, Boolean)

Restores an existing sObject record from your organization's Recycle Bin.

### Signature

```
public static Database.UndeleteResult undelete(ID recordID, Boolean opt_allOrNone)
```

### Parameters

***recordID***

Type: ID

***opt_allOrNone***

Type: Boolean

The optional *opt_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(ID[], Boolean)

Restores one or more existing sObject records, such as individual invoice statements.

### Signature

```
public static Database.UndeleteResult[] undelete(ID[] recordIDs, Boolean opt_allOrNone)
```

### Parameters

***RecordIDs***

Type: ID[]

***opt_allOrNone***

Type: Boolean

The optional *opt_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(sObject, Boolean)

Modifies an existing sObject record in your organization's data.

### Signature

```
public static Database.SaveResult update(sObject recordToUpdate, Boolean opt_allOrNone)
```

### Parameters

***recordToUpdate***

Type: sObject

***opt_allOrNone***

Type: Boolean

The optional *opt_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 `Description__c` field on a single invoice statement.

```
Invoice_Statement__c inv =
  new Invoice_Statement__c(
    Description__c='Invoice 1');
insert inv;

Invoice_Statement__c myInvoice =
  [SELECT Id, Description__c
   FROM Invoice_Statement__c
   WHERE Id = :inv.Id];
myInvoice.Description__c =
  'New description';

Database.SaveResult SR =
  Database.update(myInvoice);
```

## update(sObject [], Boolean)

Modifies one or more existing sObject records, such as individual invoice statements, in your organization's data.

### Signature

```
public static Database.SaveResult[] update(sObject[] recordsToUpdate, Boolean opt_allOrNone)
```

### Parameters

***recordsToUpdate***

Type: sObject []

***opt_allOrNone***

Type: Boolean

The optional *opt_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(sObject, Database.DmlOptions)

Modifies an existing sObject record in your organization's data.

### Signature

```
public static Database.SaveResult update(sObject recordToUpdate, Database.DmlOptions options)
```

### Parameters

***recordToUpdate***

Type: sObject

***options***

Type: Database.DMLOptions

The optional *opt_DMLOptions* parameter specifies additional data for the transaction, such as 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(sObject [], Database.DMLOptions)

Modifies one or more existing sObject records, such as individual invoice statements, in your organization's data.

### Signature

```
public static Database.SaveResult[] update(sObject[] recordsToUpdate, Database.DMLOptions options)
```

### Parameters

***recordsToUpdate***

Type: sObject []

***options***

Type: Database.DMLOptions

The optional *opt_DMLOptions* parameter specifies additional data for the transaction, such as 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(sObject, Schema.SObjectField, Boolean)

Creates a new sObject record or updates an existing sObject record within a single statement, using an optional custom field to determine the presence of existing objects.

### Signature

```
public static Database.UpsertResult upsert(sObject recordToUpsert, Schema.SObjectField external_ID_Field, Boolean opt_allOrNone)
```

### Parameters

***recordToUpsert***

Type: sObject

***external_ID_Field***

Type: Schema.SObjectField

The *External_ID_Field* 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 = Invoice_Statement__c.Fields.MyExternalId`.

***opt_allOrNone***

Type: Boolean

The optional *opt_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.

### upsert(sObject[], Schema.SObjectField, Boolean)

Creates new sObject records or updates existing sObject records within a single statement, using an optional custom field to determine the presence of existing objects.

#### Signature

```
public static Database.UpsertResult[] upsert(sObject [] recordsToUpsert, Schema.SObjectField
External_ID_Field, Boolean opt_allOrNone)
```

#### Parameters

***recordsToUpsert***

Type: sObject []

***External_ID_Field***

Type: Schema.SObjectField

The *External_ID_Field* 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 = Invoice_Statement__c.Fields.MyExternalId`.

***opt_allOrNone***

Type: Boolean

The optional *opt_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.

# Date Class

Contains methods for the Date primitive data type.

### Namespace

System

### Usage

For more information on Dates, see Primitive Data Types on page 22.

## Date Methods

The following are methods for `Date`.

***addDays(Integer)***

Adds the specified number of *`addlDays`* to a Date.

***addMonths(Integer)***

Adds the specified number of *`addlMonths`* to a Date

***addYears(Integer)***

Adds the specified number of *`addlYears`* to a Date

***day()***

Returns the day-of-month component of a Date.

***dayOfYear()***

Returns the day-of-year component of a Date.

***daysBetween(Date)***

Returns the number of days between the Date that called the method and the specified date.

***daysInMonth(Integer, Integer)***

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(Integer)***

Returns `true` if the specified year is a leap year.

***isSameDay(Date)***

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(Date)***

Returns the number of months between the Date that called the method and the specified date, ignoring the difference in dates.

***newInstance(Integer, Integer, Integer)***

Constructs a Date from Integer representations of the *year*, *month* (1=Jan), and *day*.

***parse(String)***

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(String)***

Returns a Date that contains the value of the specified String.

***valueOf(Object)***

Converts the specified history tracking field value to a Date.

***year()***

Returns the year component of a Date

## addDays(Integer)

Adds the specified number of *addlDays* to a Date.

### Signature

```
public Date addDays(Integer addlDays)
```

### Parameters

***addlDays***

Type: Integer

### Return Value

Type: Date

### Example

```
date myDate =
     date.newInstance(1960, 2, 17);
date newDate = mydate.addDays(2);
```

## addMonths(Integer)

Adds the specified number of *addlMonths* to a Date

### Signature

```
public Date addMonths(Integer addlMonths)
```

#### Parameters

***addlMonths***

Type: Integer

#### Return Value

Type: Date

### addYears(Integer)

Adds the specified number of *addlYears* to a Date

#### Signature

```
public Date addYears(Integer addlYears)
```

#### Parameters

***addlYears***

Type: Integer

#### Return Value

Type: Date

### day()

Returns the day-of-month component of a Date.

#### Signature

```
public Integer day()
```

#### Return Value

Type: Integer

#### Example

For example, February 5, 1999 would be day 5.

### dayOfYear()

Returns the day-of-year component of a Date.

#### Signature

```
public Integer dayOfYear()
```

#### Return Value

Type: Integer

#### Example

For example, February 5, 1999 would be day 36.

## daysBetween(Date)

Returns the number of days between the Date that called the method and the specified date.

### Signature

```
public Integer daysBetween(Date compDate)
```

### Parameters

***compDate***

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(Integer, Integer)

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

## isLeapYear(Integer)

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

## isSameDay(Date)

Returns `true` if the Date that called the method is the same as the specified date.

### Signature

```
public Boolean isSameDay(Date compDate)
```

### Parameters

***compDate***

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

### monthsBetween(Date)

Returns the number of months between the Date that called the method and the specified date, ignoring the difference in dates.

#### Signature

```
public Integer monthsBetween(Date compDate)
```

#### Parameters

***compDate***

Type: Date

#### Return Value

Type: Integer

#### Example

For example, March 1 and March 30 of the same year have 0 months between them.

### newInstance(Integer, Integer, Integer)

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(String)

Constructs a Date from a String. The format of the String depends on the local date format.

#### Signature

```
public static Date parse(String Date)
```

#### Parameters

***Date***

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

For example, July 14, 1999 returns July 1, 1999.

## 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(String)

Returns a Date that contains the value of the specified String.

### Signature

```
public static Date valueOf(String toDate)
```

### Parameters

***toDate***

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(Object)

Converts the specified history tracking field value to a Date.

#### 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 when the field is a Date field.

#### Example

### year()

Returns the year component of a Date

#### Signature

```
public Integer year()
```

#### Return Value

Type: Integer

## Datetime Methods

Contains methods for the Datetime primitive data type.

### Namespace

System

### Usage

For more information about the Datetime, see Primitive Data Types on page 22.

## Datetime Methods

The following are methods for `Datetime`.

***addDays(Integer)***

Adds the specified number of days to a Datetime.

***addHours(Integer)***

Adds the specified number of hours to a Datetime.

***addMinutes(Integer)***

Adds the specified number of minutes to a Datetime.

### addMonths(Integer)

Adds the specified number of months to a Datetime.

### addSeconds(Integer)

Adds the specified number of seconds to a Datetime.

### addYears(Integer)

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(String)

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(String, String)

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(String)

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(Datetime)*

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(Long)*

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(Integer, Integer, Integer)*

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), and day at midnight in the local time zone.

### *newInstance(Integer, Integer, Integer, Integer, Integer, Integer)*

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(Integer, Integer, Integer)*

Constructs a Datetime from Integer representations of the specified year, month (1=Jan), and day at midnight in the GMT time zone

### *newInstanceGmt(Integer, Integer, Integer, Integer, Integer, Integer)*

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(String)*

Constructs a Datetime from the String `datetime` 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(String)***

Returns a Datetime that contains the value of the specified string.

***valueOf(Object)***

Converts the specified history tracking field value to a Datetime.

***valueOfGmt(String)***

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(Integer)

Adds the specified number of days to a Datetime.

### Signature

```
public Datetime addDays(Integer addlDays)
```

### Parameters

***addlDays***

Type: Integer

### Return Value

Type: Datetime

### Example

```
datetime myDate =
   datetime.newInstance
      (1960, 2, 17);
datetime newDate = mydate.addDays(2);
```

## addHours(Integer)

Adds the specified number of hours to a Datetime.

#### Signature

```
public Datetime addHours(Integer addlHours)
```

#### Parameters

***addlHours***
Type: Integer

#### Return Value

Type: Datetime

### addMinutes(Integer)

Adds the specified number of minutes to a Datetime.

#### Signature

```
public Datetime addMinutes(Integer addlMinutes)
```

#### Parameters

***addlMinutes***
Type: Integer

#### Return Value

Type: Datetime

### addMonths(Integer)

Adds the specified number of months to a Datetime.

#### Signature

```
public Datetime addMonths(Integer addlMonths)
```

#### Parameters

***addlMonths***
Type: Integer

#### Return Value

Type: Datetime

### addSeconds(Integer)

Adds the specified number of seconds to a Datetime.

#### Signature

```
public Datetime addSeconds(Integer addlSeconds)
```

#### Parameters

***addlSeconds***

Type: Integer

#### Return Value

Type: Datetime

### addYears(Integer)

Adds the specified number of years to a Datetime.

#### Signature

```
public Datetime addYears(Integer addlYears)
```

#### Parameters

***addlYears***

Type: Integer

#### Return Value

Type: Datetime

### 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

### dateGMT()

Return the Date component of a Datetime in the GMT time zone.

#### Signature

```
public Date dateGMT()
```

#### Return Value

Type: Date

### 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

For example, February 5, 1999 08:30:12 would be day 5.

### dayGmt()

Returns the day-of-month component of a Datetime in the GMT time zone.

#### Signature

```
public Integer dayGmt()
```

#### Return Value

Type: Integer

#### Example

For example, February 5, 1999 08:30:12 would be day 5.

### 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

For example, February 5, 1999 08:30:12 would be day 36.

## 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

## format(String)

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 dateFormat)
```

#### Parameters

***dateFormat***

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(String, String)

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 dateFormat, String timezone)
```

#### Parameters

***dateFormat***

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(String)

Returns a Datetime as a string using the supplied Java simple date format and the GMT time zone.

#### Signature

```
public String formatGmt(String dateFormat)
```

#### Parameters

***dateFormat***

Type: String

#### Return Value

Type: String

#### Usage

For more information on the Java simple date format, see Java SimpleDateFormat.

## 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

## 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

## hourGmt()

Returns the hour component of a Datetime in the GMT time zone.

### Signature

```
public Integer hourGmt()
```

### Return Value

Type: Integer

## isSameDay(Datetime)

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 compDt)
```

### Parameters

***compDt***

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

## millisecondGmt()

Return the millisecond component of a Datetime in the GMT time zone.

### Signature

```
public Integer millisecondGmt()
```

### Return Value

Type: Integer

## 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

### minuteGmt()

Returns the minute component of a Datetime in the GMT time zone.

#### Signature

```
public Integer minuteGmt()
```

#### Return Value

Type: Integer

### 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

### monthGmt()

Returns the month component of a Datetime in the GMT time zone (1=Jan).

#### Signature

```
public Integer monthGmt()
```

#### Return Value

Type: Integer

### newInstance(Long)

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.

## newInstance(Date, Time)

Constructs a DateTime from the specified date and time in the local time zone.

### Signature

```
public static Datetime newInstance(Date dt, Time tm)
```

### Parameters

***dt***

Type: Date

***tm***

Type: Time

### Return Value

Type: Datetime

The returned date is in the GMT time zone.

## newInstance(Integer, Integer, Integer)

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(Integer, Integer, Integer, Integer, Integer, Integer)

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 dt, Time tm)
```

### Parameters

***dt***

Type: Date

***tm***

Type: Time

#### Return Value

Type: Datetime

### newInstanceGmt(Integer, Integer, Integer)

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

### newInstanceGmt(Integer, Integer, Integer, Integer, Integer, Integer)

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

### 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(String)

Constructs a Datetime from the String *datetime* in the local time zone and in the format of the user locale.

#### Signature

```
public static Datetime parse(String datetime)
```

#### Parameters

***datetime***

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

## secondGmt()

Returns the second component of a Datetime in the GMT time zone.

### Signature

```
public Integer secondGmt()
```

### Return Value

Type: Integer

## 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

## timeGmt()

Returns the time component of a Datetime in the GMT time zone.

### Signature

```
public Time timeGmt()
```

### Return Value

Type: Time

## valueOf(String)

Returns a Datetime that contains the value of the specified string.

### Signature

```
public static Datetime valueOf(String toDateTime)
```

#### Parameters

***toDateTime***

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(Object)

Converts the specified history tracking field value to a Datetime.

#### 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 when the field is a Date/Time field.

#### Example

### valueOfGmt(String)

Returns a Datetime that contains the value of the specified String.

#### Signature

```
public static Datetime valueOfGmt(String toDateTime)
```

#### Parameters

***toDateTime***

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.

### 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

### yearGmt()

Returns the year component of a Datetime in the GMT time zone.

#### Signature

```
public Integer yearGmt()
```

#### Return Value

Type: Integer

## Decimal Class

Contains methods for the Decimal primitive data type.

### Namespace

System

### Usage

For more information on Decimal, see Primitive Data Types on page 22.

#### *Rounding Mode*

Rounding mode specifies the rounding behavior for numerical operations capable of discarding precision.

#### *Decimal Methods*

## 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:<br>• Input number 5.5: CEILING round mode result: 6<br>• Input number 1.1: CEILING round mode result: 2<br>• Input number -1.1: CEILING round mode result: -1<br>• Input number -2.7: CEILING round mode result: -2 |
| 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:<br>• Input number 5.5: DOWN round mode result: 5<br>• Input number 1.1: DOWN round mode result: 1<br>• Input number -1.1: DOWN round mode result: -1<br>• Input number -2.7: DOWN round mode result: -2 |
| 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:<br>• Input number 5.5: FLOOR round mode result: 5<br>• Input number 1.1: FLOOR round mode result: 1<br>• Input number -1.1: FLOOR round mode result: -2<br>• Input number -2.7: FLOOR round mode result: -3 |
| 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:<br>• Input number 5.5: HALF_DOWN round mode result: 5<br>• Input number 1.1: HALF_DOWN round mode result: 1<br>• Input number -1.1: HALF_DOWN round mode result: -1<br>• Input number -2.7: HALF_DOWN round mode result: -2 |
| 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 |

| Name | Description |
|---|---|
| | • Input number -1.1: `HALF_EVEN` round mode result: -1<br>• Input number -2.7: `HALF_EVEN` round mode result: -3<br><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 |
| `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, an Exception is thrown. For example:<br>• Input number 5.5: `UNNECESSARY` round mode result: Exception<br>• Input number 1.0: `UNNECESSARY` round mode result: 1 |
| `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:<br>• Input number 5.5: `UP` round mode result: 6<br>• Input number 1.1: `UP` round mode result: 2<br>• Input number -1.1: `UP` round mode result: -2<br>• Input number -2.7: `UP` round mode result: -3 |

## Decimal Methods

The following are methods for `Decimal`.

### *abs()*

Returns the absolute value of the Decimal.

### *divide(Decimal, Integer)*

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.

### *divide(Decimal, Integer, Object)*

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.

### *doubleValue()*

Returns the Double value of this Decimal.

### *format()*

Returns the String value of this Decimal using the locale of the context user.

### *intValue()*

Returns the Integer value of this Decimal.

### *longValue()*

Returns the Long value of this Decimal.

### *pow(Integer)*

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(System.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(Integer)*

Sets the scale of the Decimal to the given number of decimal places, using half-even rounding, if necessary. Half-even rounding mode rounds towards the “nearest neighbor” unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.

### *setScale(Integer, System.RoundingMode)*

Sets the scale of the Decimal to the given number of decimal places, using the rounding mode specified by the 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(Double)*

Returns a Decimal that contains the value of the specified Double.

### *valueOf(Long)*

Returns a Decimal that contains the value of the specified Long.

### *valueOf(String)*

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

## divide(Decimal, Integer)

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

In the following example, D has the value of 0.190.

```
Decimal D = 19;

D.Divide(100, 3);
```

## divide(Decimal, Integer, Object)

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 on page 763

#### 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

### 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.

### intValue()

Returns the Integer value of this Decimal.

#### Signature

```
public Integer intValue()
```

#### Return Value

Type: Integer

### longValue()

Returns the Long value of this Decimal.

#### Signature

```
public Long longValue()
```

#### Return Value

Type: Long

### pow(Integer)

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(System.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

### setScale(Integer)

Sets the scale of the Decimal to the given number of decimal places, using half-even rounding, if necessary. Half-even rounding mode rounds towards the “nearest neighbor” unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.

#### Signature

```
public Decimal setScale(Integer scale)
```

#### Parameters

***scale***

Type: Integer

The value of *scale* must be between –33 and 33.

#### Return Value

Type: Decimal

#### Usage

If you do not explicitly set the scale for a Decimal, the scale is determined by the item from which the Decimal is created:

- 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 scale is determined by converting the number to a String and then using the number of characters after the decimal point.

### setScale(Integer, System.RoundingMode)

Sets the scale of the Decimal to the given number of decimal places, using the rounding mode specified by the rounding mode, if necessary.

#### Signature

```
public Decimal setScale(Integer scale, System.RoundingMode roundingMode)
```

#### Parameters

***scale***

Type: Integer

The value of *scale* must be between -32,768 and 32,767.

***roundingMode***

Type: System.RoundingMode

#### Return Value

Type: Decimal

#### Usage

If you do not explicitly set the scale for a Decimal, the scale is determined by the item from which the Decimal is created:

- 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 scale is determined by converting the number to a String and then 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

### toPlainString()

Returns the String value of this Decimal, without using scientific notation.

#### Signature

```
public String toPlainString()
```

#### Return Value

Type: String

### valueOf(Double)

Returns a Decimal that contains the value of the specified Double.

#### Signature

```
public static Decimal valueOf(Double convertToDecimal)
```

#### Parameters

***convertToDecimal***
Type: Double

#### Return Value

Type: Decimal

### valueOf(Long)

Returns a Decimal that contains the value of the specified Long.

#### Signature

```
public static Decimal valueOf(Long convertToDecimal)
```

#### Parameters

***convertToDecimal***
Type: Long

#### Return Value

Type: Decimal

### valueOf(String)

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 convertToDecimal)
```

#### Parameters

***convertToDecimal***
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 22.

## Double Methods

The following are methods for `Double`.

***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(String)***

Returns a Double that contains the value of the specified String. As in Java, the String is interpreted as representing a signed decimal.

***valueOf(Object)***

Converts the specified history tracking field value to 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

### 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

## 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(String)

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 toDouble)
```

#### Parameters

***toDouble***

Type: String

#### Return Value

Type: Double

#### Example

```
Double DD1 = double.valueOf('3.14159');
```

### valueOf(Object)

Converts the specified history tracking field value to 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 when the field type corresponds to a Double type, like a number field.

#### Example

## 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 Database.com. You can, however, download a document from Database.com. 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.

#### *base64Decode(String)*

Converts a Base64-encoded String to a Blob representing its normal form.

***base64Encode(Blob)***

Converts a Blob to an unencoded String representing its normal form.

***convertFromHex(String)***

Converts the specified hexadecimal (base 16) string to a Blob value and returns this Blob value.

***convertToHex(Blob)***

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).

***urlDecode(String, String)***

Decodes a string in `application/x-www-form-urlencoded` format using a specific encoding scheme, for example "UTF-8."

***urlEncode(String, String)***

Encodes a string into the `application/x-www-form-urlencoded` format using a specific encoding scheme, for example "UTF-8."

## base64Decode(String)

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(Blob)

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

## convertFromHex(String)

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(Blob)

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(String, String)

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(String, String)

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 29.

## 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. |
| `InvalidParameterValueException` | An invalid parameter was supplied for a method. |
| `JSONException` | Any problem with JSON serialization and deserialization operations. For more information, see the methods of `System.JSON`, `System.JSONParser`, and `System.JSONGenerator`. |

| Exception | Description |
|---|---|
| `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. |
| `NoSuchElementException` | Used specifically by the Iterator `next` method. This exception is thrown if you try to access items beyond the end of the list. For example, if `iterator.hasNext() == false` and you call `iterator.next()`, this exception is thrown. |
| `NullPointerException` | Any problem with dereferencing null, such as in the following code:<br>`String s;`<br>`s.toLowerCase(); // Since s is null, this call causes`<br>`// 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. |
| `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. |
| `XmlException` | Any problem with the XmlStream classes, such as failing to read or write XML. |

The following is an example using the DmlException exception:

```
Invoice_Statement__c[] invs = new Invoice_Statement__c[]{
   new Invoice_Statement__c(Description__c = 'Invoice 1')};
try {
    insert invs;
} 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:

- `ConnectApi` 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. |
| `getDmlType` | Integer *i* | System.StatusCode | Returns the value of the System.StatusCode enum. For example:<br>`try { insert new Invoice_Statement__c(); } catch (System.DmlException ex) { System.assertEquals( StatusCode.REQUIRED_FIELD_MISSING, ex.getDmlType(0)); }`<br>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.

***send(HttpRequest)***

Sends an HttpRequest and returns the response.

***toString()***

Returns a string that displays and identifies the object's properties.

### send(HttpRequest)

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`.

***respond(HttpRequest)***

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.

### respond(HttpRequest)

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 req)
```

#### Parameters

***req***

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());
  }
}
```

### 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.

***HttpRequest Constructors***
***HttpRequest Methods***

**See Also:**

*JSON Support*
*XML Support*

## HttpRequest Constructors

The following are constructors for `HttpRequest`.

***HttpRequest()***

Creates a new instance of the `HttpRequest` class.

### HttpRequest()

Creates a new instance of the `HttpRequest` class.

#### Signature

```
public HttpRequest()
```

## HttpRequest Methods

The following are methods for `HttpRequest`. All are instance methods.

### *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(String)*

Retrieves the contents of the request header.

### *getMethod()*

Returns the type of method used by `HttpRequest`.

### *setBody(String)*

Sets the contents of the body for this request.

### *setBodyAsBlob(Blob)*

Sets the contents of the body for this request using a Blob.

### *setBodyDocument(Dom.Document)*

Sets the contents of the body for this request. The contents represent a DOM document.

### *setClientCertificate(String, String)*

This method is deprecated. Use `setClientCertificateName` instead.

### *setClientCertificateName(String)*

If the external service requires a client certificate for authentication, set the certificate name.

### *setCompressed(Boolean)*

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(String)*

Sets the URL for the endpoint of the external server for this request.

### *setHeader(String, String)*

Sets the contents of the request header.

### *setMethod(String)*

Sets the type of method to be used for the HTTP request.

### *setTimeout(Integer)*

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(String)

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(String)

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: 3 MB.

The HTTP request and response sizes count towards the total heap size.

## setBodyAsBlob(Blob)

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: 3 MB.

The HTTP request and response sizes count towards the total heap size.

## setBodyDocument(Dom.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: 3 MB.

### setClientCertificate(String, String)

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(String)

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(Boolean)

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(String)

Sets the URL for the endpoint of the external server for this request.

### Signature

```
public Void setEndpoint(String endpoint)
```

### Parameters

***endpoint***

Type: String

### Return Value

Type: Void

## setHeader(String, String)

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(String)

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(Integer)

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('http://www.cheenath.com/tutorial/sample1/build.xml');
    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.

***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(String)***

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(String)***

Specifies the body returned in the response.

***setBodyAsBlob(Blob)***

Specifies the body returned in the response using a Blob.

***setHeader(String, String)***

Specifies the contents of the response header.

***setStatus(String)***

Specifies the status message returned in the response.

***setStatusCode(Integer)***

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

Limit3 MB. 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

Limit3 MB. 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(String)

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(String)

Specifies the body returned in the response.

#### Signature

```
public Void setBody(String body)
```

#### Parameters

***body***

Type: String

#### Return Value

Type: Void

### setBodyAsBlob(Blob)

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(String, String)

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(String)

Specifies the status message returned in the response.

#### Signature

```
public Void setStatus(String status)
```

#### Parameters

***status***
Type: String

#### Return Value

Type: Void

### setStatusCode(Integer)

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);
        System.debug(queryString);
        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`.

***addError(String)***

Marks a record with a custom error message and prevents any DML operation from occurring.

***addError(Exception)***

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(String)***

Converts the specified String into an ID and returns the ID.

### addError(String)

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(String) sObject method.

#### Example

```
Trigger.new[0].Id.addError('bad');
```

### addError(Exception)

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(Exception) sObject method.

### 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.

### valueOf(String)

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

## 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 Database.com 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.

***findSimilar(Idea)***

Returns a list similar ideas based on the title of the specified idea.

***getAllRecentReplies(String, String)***

Returns ideas that have recent replies for the specified user or zone. This includes all read and unread replies.

***getReadRecentReplies(String, String)***

Returns ideas that have recent replies marked as read.

***getUnreadRecentReplies(String, String)***

Returns ideas that have recent replies marked as unread.

***markRead(String)***

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(String, String)

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(String, String)

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(String, String)

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(String)

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

## Integer Class

Contains methods for the Integer primitive data type.

### Namespace

System

### Usage

For more information on integers, see Primitive Data Types on page 22.

## Integer Methods

The following are methods for `Integer`.

***format()***

Returns the integer as a string using the locale of the context user.

***valueOf(String)***

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(Object)***

Converts the specified history tracking field value to 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

### valueOf(String)

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 toInteger)
```

#### Parameters

***toInteger***

Type: String

#### Return Value

Type: Integer

#### Example

```
Integer myInt = Integer.valueOf('123');
```

### valueOf(Object)

Converts the specified history tracking field value to 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 when the field type corresponds to an Integer type, like a number field.

#### Example

## 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.

## JSON Methods

The following are methods for `JSON`. All methods are static.

***createGenerator(Boolean)***

Returns a new JSON generator.

***createParser(String)***

Returns a new JSON parser.

***deserialize(String, System.Type)***

Deserializes the specified JSON string into an Apex object of the specified type.

***deserializeStrict(String, System.Type)***

Deserializes the specified JSON string into an Apex object of the specified type.

***deserializeUntyped(String)***

Deserializes the specified JSON string into collections of primitive data types.

***serialize(Object)***

Serializes Apex objects into JSON content.

***serializePretty(Object)***

Serializes Apex objects into JSON content and generates indented content using the pretty-print format.

## createGenerator(Boolean)

Returns a new JSON generator.

### Signature

```
public static System.JSONGenerator createGenerator(Boolean pretty)
```

### Parameters

***pretty***

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(String)

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(String, System.Type)

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.com 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(String, System.Type)

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(String)

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(Object)

Serializes Apex objects into JSON content.

#### Signature

```
public static String serialize(anyType object)
```

#### Parameters

***anyType***

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(Object)

Serializes Apex objects into JSON content and generates indented content using the pretty-print format.

#### Signature

```
public static String serializePretty(Object anyType)
```

#### Parameters

***anyType***

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.

## JSONGenerator Methods

The following are methods for `JSONGenerator`. All are instance methods.

***close()***

Closes the JSON generator.

***getAsString()***

Returns the generated JSON content.

***isClosed()***

Returns `true` if the JSON generator is closed; otherwise, returns `false`.

***writeBlob(Blob)***

Writes the specified `Blob` value as a base64-encoded string.

***writeBlobField(String, Blob)***

Writes a field name and value pair using the specified field name and BLOB value.

***writeBoolean(Boolean)***

Writes the specified Boolean value.

***writeBooleanField(String, Boolean)***

Writes a field name and value pair using the specified field name and Boolean value.

***writeDate(Date)***

Writes the specified date value in the ISO-8601 format.

***writeDateField(String, Date)***

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(Datetime)***

Writes the specified date and time value in the ISO-8601 format.

***writeDateTimeField(String, Datetime)***

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(String)***

Writes a field name.

***writeId(ID)***

Writes the specified ID value.

***writeIdField(String, Id)***

Writes a field name and value pair using the specified field name and identifier value.

***writeNull()***

Writes the JSON null literal value.

***writeNullField(String)***

Writes a field name and value pair using the specified field name and the JSON null literal value.

***writeNumber(Decimal)***

Writes the specified decimal value.

***writeNumber(Double)***

Writes the specified double value.

***writeNumber(Integer)***

Writes the specified integer value.

***writeNumber(Long)***

Writes the specified long value.

***writeNumberField(String, Decimal)***

Writes a field name and value pair using the specified field name and decimal value.

***writeNumberField(String, Double)***

Writes a field name and value pair using the specified field name and double value.

***writeNumberField(String, Integer)***

Writes a field name and value pair using the specified field name and integer value.

***writeNumberField(String, Long)***

Writes a field name and value pair using the specified field name and long value.

***writeObject(Any type)***

Writes the specified Apex object in JSON format.

***writeObjectField(String, Any type)***

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(String)***

Writes the specified string value.

***writeStringField(String, String)***

Writes a field name and value pair using the specified field name and string value.

***writeTime(Time)***

Writes the specified time value in the ISO-8601 format.

***writeTimeField(String, Time)***

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(Blob)

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(String, Blob)

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(Boolean)

Writes the specified Boolean value.

#### Signature

```
public Void writeBoolean(Boolean blobValue)
```

#### Parameters

***blobValue***

Type: Boolean

#### Return Value

Type: Void

### writeBooleanField(String, Boolean)

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(Date)

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(String, Date)

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(Datetime)

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(String, Datetime)

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(String)

Writes a field name.

#### Signature

```
public Void writeFieldName(String fieldName)
```

#### Parameters

***fieldName***
Type: String

#### Return Value

Type: Void

### writeId(ID)

Writes the specified ID value.

#### Signature

```
public Void writeId(ID identifier)
```

#### Parameters

***identifier***
Type: ID

#### Return Value

Type: Void

### writeIdField(String, Id)

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(String)

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(Decimal)

Writes the specified decimal value.

#### Signature

```
public Void writeNumber(Decimal number)
```

#### Parameters

***number***

Type: Decimal

#### Return Value

Type: Void

### writeNumber(Double)

Writes the specified double value.

#### Signature

```
public Void writeNumber(Double number)
```

#### Parameters

***number***

Type: Double

#### Return Value

Type: Void

## writeNumber(Integer)

Writes the specified integer value.

### Signature

```
public Void writeNumber(Integer number)
```

### Parameters

***number***

Type: Integer

### Return Value

Type: Void

## writeNumber(Long)

Writes the specified long value.

### Signature

```
public Void writeNumber(Long number)
```

### Parameters

***number***

Type: Long

### Return Value

Type: Void

## writeNumberField(String, Decimal)

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(String, Double)

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(String, Integer)

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(String, Long)

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(Any type)

Writes the specified Apex object in JSON format.

#### Signature

```
public Void writeObject(Object anyType)
```

#### Parameters

***anyType***
Type: Object

#### Return Value

Type: Void

### writeObjectField(String, Any type)

Writes a field name and value pair using the specified field name and Apex object.

#### Signature

```
public Void writeObjectField(String fieldName, Object anyType)
```

#### Parameters

***fieldName***
Type: String

***anyType***
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(String)

Writes the specified string value.

#### Signature

```
public Void writeString(String stringValue)
```

#### Parameters

***stringValue***
Type: String

#### Return Value

Type: Void

### writeStringField(String, String)

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(Time)

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(String, Time)

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.

## JSONParser Methods

The following are methods for `JSONParser`. All are instance methods.

***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(System.Type)***

Deserializes JSON content into an object of the specified Apex type and returns the deserialized object.

***readValueAsStrict(System.Type)***

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(System.Type)

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.com 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(System.Type)

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. |

| Enum Value | Description |
|---|---|
| VALUE_NULL | The literal “null” value. |
| VALUE_NUMBER_FLOAT | A float value. |
| VALUE_NUMBER_INT | An integer value. |
| 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 Understanding Execution Governors and Limits on page 203.

# Limits Methods

The following are methods for `Limits`. All methods are static.

### *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()*

Returns the number of child relationship objects that have been returned.

### getLimitChildRelationshipsDescribes()

Returns the total number of child relationship objects that can be returned.

### getCpuTime()

Returns the CPU time (in milliseconds) accumulated on the Database.com 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.

### getFieldsDescribes()

Returns the number of field describe calls that have been made.

### getLimitFieldsDescribes()

Returns the total number of field describe calls that can be made.

### 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.

### getQueries()

Returns the number of SOQL queries that have been issued.

### getLimitQueries()

Returns the total number of SOQL queries that can be issued.

### getPicklistDescribes()

Returns the number of PicklistEntry objects that have been returned.

### getLimitPicklistDescribes()

Returns the total number of PicklistEntry objects that can be returned.

***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.

***getRecordTypesDescribes()***

Returns the number of RecordTypeInfo objects that have been returned.

***getLimitRecordTypesDescribes()***

Returns the total 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()

Returns the number of child relationship objects that have been returned.

#### Signature

```
public static Integer getChildRelationshipsDescribes()
```

#### Return Value

Type: Integer

### getLimitChildRelationshipsDescribes()

Returns the total number of child relationship objects that can be returned.

#### Signature

```
public static Integer getLimitChildRelationshipsDescribes()
```

#### Return Value

Type: Integer

### getCpuTime()

Returns the CPU time (in milliseconds) accumulated on the Database.com 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

## getFieldsDescribes()

Returns the number of field describe calls that have been made.

### Signature

```
public static Integer getFieldsDescribes()
```

### Return Value

Type: Integer

## getLimitFieldsDescribes()

Returns the total number of field describe calls that can be made.

### Signature

```
public static Integer getLimitFieldsDescribes()
```

### 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

### 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

### getPicklistDescribes()

Returns the number of PicklistEntry objects that have been returned.

#### Signature

```
public static Integer getPicklistDescribes()
```

#### Return Value

Type: Integer

### getLimitPicklistDescribes()

Returns the total number of PicklistEntry objects that can be returned.

#### Signature

```
public static Integer getLimitPicklistDescribes()
```

#### 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

### getRecordTypesDescribes()

Returns the number of RecordTypeInfo objects that have been returned.

#### Signature

```
public static Integer getRecordTypesDescribes()
```

#### Return Value

Type: Integer

### getLimitRecordTypesDescribes()

Returns the total number of RecordTypeInfo objects that can be returned.

#### Signature

```
public static Integer getLimitRecordTypesDescribes()
```

#### Return Value

Type: Integer

### 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

Because Apex statements are no longer counted, this method returns a value that resembles statement usage but is only an approximation. 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.

**Note:** Because Apex statements are no longer counted, do not use this method. Call getCpuTime() instead.

## getLimitScriptStatements()

Deprecated. Returns the maximum number of Apex statements that can execute.

### Signature

```
public static Integer getLimitScriptStatements()
```

### Return Value

Type: Integer

### Usage

**Note:** Because Apex statements are no longer counted, do not use this method. 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 25.

***List Constructors***
***List Methods***

## List Constructors

The following are constructors for `List`.

#### *List<T>()*

Creates a new instance of the `List` class. A list can hold elements of any data type T.

***List<T>(List<T>)***

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.

***List<T>(Set<T>)***

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.

## 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>(List<T>)

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>(Set<T>)

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.

***add(Object)***

Adds an element to the end of the list.

***add(Integer, Object)***

Inserts an element into the list at the specified index position.

***addAll(List)***

Adds all of the elements in the specified list to the list that calls the method. Both lists must be of the same type.

***addAll(Set)***

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.

***clear()***

Removes all elements from a list, consequently setting the list's length to zero.

***clone()***

Makes a duplicate copy of a list.

***deepClone(Boolean, Boolean, Boolean)***

Makes a duplicate copy of a list of sObject records, including the sObject records themselves.

***equals(List)***

Compares this list with the specified list and returns `true` if both lists are equal; otherwise, returns `false`.

***get(Integer)***

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(Integer)***

Removes the list element stored at the specified index, returning the element that was removed.

***set(Integer, Object)***

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(Object)

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(myNumber, 47);
```

## add(Integer, Object)

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(myList.get(1), 52);
```

### addAll(List)

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(Set)

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

```
Invoice_Statement__c a = new
   Invoice_Statement__c(
    Description__c='Invoice1');

Invoice_Statement__c b = new
     Invoice_Statement__c();

Invoice_Statement__c[] q1 = new
     Invoice_Statement__c[]{a,b};

Invoice_Statement__c[] q2 =
   q1.clone();
q1[0].Description__c =
   'New description';

System.assertEquals(
      q1[0].Description__c,
      'New description');
System.assertEquals(
      q2[0].Description__c,
      'New description');
```

## deepClone(Boolean, Boolean, Boolean)

Makes a duplicate copy of a list of sObject records, including the sObject records themselves.

### Signature

```
public List<Object> deepClone(Boolean opt_preserve_id, Boolean
opt_preserve_readonly_timestamps, Boolean opt_preserve_autonumber)
```

### Parameters

***opt_preserve_id***

Type: Boolean

The optional *opt_preserve_id* 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.

***opt_preserve_readonly_timestamps***

Type: Boolean

The optional *opt_preserve_readonly_timestamps* 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.

***opt_preserve_autonumber***

Type: Boolean

The optional *opt_preserve_autonumber* 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 Salesforce.comAPI version 22.0 or earlier, the default value for the *opt_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 invoice statements.

```
Invoice_Statement__c a = new
   Invoice_Statement__c(
      Description__c='Invoice1');

Invoice_Statement__c b =
    new Invoice_Statement__c();

Invoice_Statement__c[] q1 = new
   Invoice_Statement__c[]{a,b};

Invoice_Statement__c[] q2 =
```

```
   q1.deepClone();
q1[0].Description__c = 'New description';

System.assertEquals(
     q1[0].Description__c,
     'New description');

System.assertEquals(
     q2[0].Description__c,
     'Invoice1');
```

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<Invoice_Statement__c> invs =
    [SELECT CreatedById,
    CreatedDate, LastModifiedById,
    LastModifiedDate, Description__c
    FROM Invoice_Statement__c
    WHERE Id = :a.Id OR Id = :b.Id];

// Clone list while preserving
// timestamp and user ID fields.
Invoice_Statement__c[] q3 =
   invs.deepClone(false,true,false);

// Verify timestamp fields are
// preserved for the first
// list element.
System.assertEquals(
    q3[0].CreatedById,
    invs[0].CreatedById);
System.assertEquals(
    q3[0].CreatedDate,
    invs[0].CreatedDate);
System.assertEquals(
    q3[0].LastModifiedById,
    invs[0].LastModifiedById);
System.assertEquals(
    q3[0].LastModifiedDate,
    invs[0].LastModifiedDate);
```

## equals(List)

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(Integer)

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(myNumber, 47);
```

```
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 131.

#### Example

```
Invoice_Statement__c a =
   new Invoice_Statement__c();
insert a;
// Create a generic sObject
// variable s
SObject s = Database.query
  ('SELECT Id FROM ' +
   'Invoice_Statement__c ' +
   'LIMIT 1');

// Verify if that sObject
// variable is
// an invoice statement token
System.assertEquals(
s.getSObjectType(),
Invoice_Statement__c.sObjectType);

// Create a list of generic sObjects
List<sObject> q =
   new Invoice_Statement__c[]{};

// Verify if the list of sObjects
// contains invoice statement tokens
System.assertEquals(
q.getSObjectType(),
Invoice_Statement__c.sObjectType);
```

## 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<Invoice_Statement__c>{

   List<Invoice_Statement__c>
      invoices {get; set;}
   Integer i {get; set;}

 public CustomIterable(){
  invoices =
  [SELECT Id, Description__c
  FROM Invoice_Statement__c
  WHERE Description__c = 'false'];

       i = 0;
   }

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

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

## remove(Integer)

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(S1, 'Green');
```

### set(Integer, Object)

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(myList.get(1), 52);
```

```
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(size, 0);

List<Integer> myList2 = new Integer[6];
Integer size2 = myList2.size();
system.assertEquals(size2, 6);
```

## 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, 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(q1.get(1), 5);
```

# Long Class

Contains methods for the Long primitive data type.

### Namespace

System

### Usage

For more information on Long, see Primitive Data Types on page 22.

## Long Methods

The following are methods for `Long`.

***format()***

Returns the String format for this Long using the locale of the context user.

***intValue()***

Returns the Integer value for this Long.

***valueOf(String)***

Returns a Long that contains the value of the specified String. As in Java, the string is interpreted as representing a signed decimal Long.

### format()

Returns the String format for this Long using the locale of the context user.

#### Signature

```
public String format()
```

#### Return Value

Type: String

### intValue()

Returns the Integer value for this Long.

#### Signature

```
public Integer intValue()
```

#### Return Value

Type: Integer

### valueOf(String)

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 toLong)
```

### Parameters

***toLong***

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 27.



## Map Constructors

The following are constructors for `Map`.

### *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.

### *Map<T1, T2>(Map<T1, T2>)*

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.

### *Map<ID, sObject>(List<sObject>)*

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.

## 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>(Map<T1, T2>)

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>(List<sObject>)

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.

***clear()***

Removes all of the key-value mappings from the map.

***clone()***

Makes a duplicate copy of the map.

***containsKey(Object)***

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(Map)***

Compares this map with the specified map and returns `true` if both maps are equal; otherwise, returns `false`.

***get(Object)***

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(Object, Object)***

Associates the specified value with the specified key in the map.

***putAll(Map)***

Copies all of the mappings from the specified map to the original map.

***putAll(sObject[])***

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 of the values in the map in arbitrary order.

## 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

```
Invoice_Statement__c a = new
   Invoice_Statement__c(
     Description__c='Invoice1');

Map<Integer,
  Invoice_Statement__c> map1 =
    new Map<Integer, Invoice_Statement__c> {};
map1.put(1, a);

Map<Integer,
```

```
   Invoice_Statement__c> map2 =
     map1.clone();
map1.get(1).Description__c =
   'New invoice';

System.assertEquals(
   map1.get(1).Description__c,
   'New invoice');

System.assertEquals(
   map2.get(1).Description__c,
   'New invoice');
```

## containsKey(Object)

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(contains, True);
```

## 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

```
Invoice_Statement__c a = new
   Invoice_Statement__c(
     Description__c='Invoice1');

Map<Integer,
   Invoice_Statement__c> map1 =
      new Map<Integer,
            Invoice_Statement__c> {};

map1.put(1, a);

Map<Integer,
   Invoice_Statement__c> map2 =
      map1.deepClone();

map1.get(1).Description__c =
   'New invoice';

System.assertEquals(
   map1.get(1).
   Description__c, 'New invoice');

System.assertEquals(
   map2.get(1).
   Description__c, 'Invoice1');
```

## equals(Map)

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(Object)

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(code, '0000A0');

// The following is not a color
// in the map
String code2 =
   colorCodes.get('Magenta');

System.assertEquals(code2, null);
```

## 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 131.

### Example

```
Invoice_Statement__c a = new
   Invoice_Statement__c(
      Description__c='Invoice1');
insert a;

// Create a generic sObject
// variable s
SObject s = Database.query
   ('SELECT Id FROM ' +
    'Invoice_Statement__c ' +
    'LIMIT 1');

// Verify if that sObject
// variable is an
// Invoice_Statement__c token
System.assertEquals(
   s.getSObjectType(),
   Invoice_Statement__c.sObjectType);

// Create a map of generic
// sObjects
Map<Integer,
   Invoice_Statement__c> M =
     new Map<Integer,
       Invoice_Statement__c>();

// Verify if the list of
// sObjects contains
// Invoice_Statement__c tokens
System.assertEquals(
   M.getSObjectType(),
   Invoice_Statement__c.sObjectType);
```

## 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(empty, true);
```

## 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(Object, Object)

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(Map)

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.

## putAll(sObject[])

Adds the list of sObject records to a map declared as Map<ID, sObject> or Map<String, sObject>.

#### Signature

```
public putAll(sObject[] sobjectArray)
```

#### Parameters

***sobjectArray***

Type: sObject[]

#### Return Value

Type:

#### Usage

This method is similar to calling the Map constructor with the same input.

## 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(code2, null);
```

### 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(mSize, 2);
```

### values()

Returns a list that contains all of the values in the map in arbitrary order.

### Signature

```
public List<Object> values()
```

### Return Value

Type: List<Object>

### 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`.

***end()***

Returns the position after the last matched character.

***end(Integer)***

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(Integer)***

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(Integer)***

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(String)*

Returns a literal replacement string for the specified string `inputString`. The characters in the returned string match the sequence of characters in `inputString`.

### *region(Integer, Integer)*

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(String)*

Replaces every subsequence of the input sequence that matches the pattern with the replacement string.

### *replaceFirst(String)*

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(String)*

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(Integer)***

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(Boolean)***

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(Boolean)***

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(Integer)

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(Integer)

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(Integer)

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(String)

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(Integer, Integer)

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(String)

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(String)

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(String)

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(Integer)

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 266.

## useAnchoringBounds(Boolean)

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 266.

### 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(Boolean)

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 Methods

The following are methods for `Math`. All methods are static.

### *abs(Decimal)*

Returns the absolute value of the specified Decimal.

### *abs(Double)*

Returns the absolute value of the specified Double.

### *abs(Integer)*

Returns the absolute value of the specified Integer.

### *abs(Long)*

Returns the absolute value of the specified Long.

### *acos(Decimal)*

Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

### *acos(Double)*

Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

### *asin(Decimal)*

Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.

### *asin(Double)*

Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.

### *atan(Decimal)*

Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

### *atan(Double)*

Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

### *atan2(Decimal, Decimal)*

Converts rectangular coordinates (*x* and *y*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *x*/*y* in the range of -*pi* to *pi*.

### *atan2(Double, Double)*

Converts rectangular coordinates (*x* and *y*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *x*/*y* in the range of -*pi* to *pi*.

### *cbrt(Decimal)*

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(Double)*

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(Decimal)

Returns the smallest (closest to negative infinity) Decimal that is not less than the argument and is equal to a mathematical integer.

### ceil(Double)

Returns the smallest (closest to negative infinity) Double that is not less than the argument and is equal to a mathematical integer.

### cos(Decimal)

Returns the trigonometric cosine of the angle specified by *d*.

### cos(Double)

Returns the trigonometric cosine of the angle specified by *d*.

### cosh(Decimal)

Returns the hyperbolic cosine of *d*. The hyperbolic cosine of *d* is defined to be $(e^x + e^{-x})/2$ where *e* is Euler's number.

### cosh(Double)

Returns the hyperbolic cosine of *d*. The hyperbolic cosine of *d* is defined to be $(e^x + e^{-x})/2$ where *e* is Euler's number.

### exp(Decimal)

Returns Euler's number *e* raised to the power of the specified Decimal.

### exp(Double)

Returns Euler's number *e* raised to the power of the specified Double.

### floor(Decimal)

Returns the largest (closest to positive infinity) Decimal that is not greater than the argument and is equal to a mathematical integer.

### floor(Double)

Returns the largest (closest to positive infinity) Double that is not greater than the argument and is equal to a mathematical integer.

### log(Decimal)

Returns the natural logarithm (base *e*) of the specified Decimal.

### log(Double)

Returns the natural logarithm (base *e*) of the specified Double.

### log10(Decimal)

Returns the logarithm (base *10*) of the specified Decimal.

### log10(Double)

Returns the logarithm (base *10*) of the specified Double.

### max(Decimal, Decimal)

Returns the larger of the two specified Decimals.

### max(Double, Double)

Returns the larger of the two specified Doubles.

### max(Integer, Integer)

Returns the larger of the two specified Integers.

### max(Long, Long)

Returns the larger of the two specified Longs.

### min(Decimal, Decimal)

Returns the smaller of the two specified Decimals.

### min(Double, Double)

Returns the smaller of the two specified Doubles.

### min(Integer, Integer)

Returns the smaller of the two specified Integers.

### min(Long, Long)

Returns the smaller of the two specified Longs.

### mod(Integer, Integer)

Returns the remainder of `i1` divided by `i2`.

### mod(Long, Long)

Returns the remainder of `L1` divided by `L2`.

### pow(Double, Double)

Returns the value of the first Double raised to the power of `exp`.

### random()

Returns a positive Double that is greater than or equal to 0.0 and less than 1.0.

### rint(Decimal)

Returns the value that is closest in value to `d` and is equal to a mathematical integer.

### rint(Double)

Returns the value that is closest in value to `d` and is equal to a mathematical integer.

### round(Double)

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(Decimal)

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(Decimal)

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(Double)

Returns the closest Long to the specified Double.

***signum(Decimal)***

Returns the signum function of the specified Decimal, which is 0 if *d* is 0, 1.0 if *d* is greater than 0, -1.0 if *d* is less than 0.

***signum(Double)***

Returns the signum function of the specified Double, which is 0 if *d* is 0, 1.0 if *d* is greater than 0, -1.0 if *d* is less than 0.

***sin(Decimal)***

Returns the trigonometric sine of the angle specified by *d*.

***sin(Double)***

Returns the trigonometric sine of the angle specified by *d*.

***sinh(Decimal)***

Returns the hyperbolic sine of *d*. The hyperbolic sine of *d* is defined to be $(e^x - e^{-x})/2$ where *e* is Euler's number.

***sinh(Double)***

Returns the hyperbolic sine of *d*. The hyperbolic sine of *d* is defined to be $(e^x - e^{-x})/2$ where *e* is Euler's number.

***sqrt(Decimal)***

Returns the correctly rounded positive square root of *d*.

***sqrt(Double)***

Returns the correctly rounded positive square root of *d*.

***tan(Decimal)***

Returns the trigonometric tangent of the angle specified by *d*.

***tan(Double)***

Returns the trigonometric tangent of the angle specified by *d*.

***tanh(Decimal)***

Returns the hyperbolic tangent of *d*. The hyperbolic tangent of *d* 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(Double)***

Returns the hyperbolic tangent of *d*. The hyperbolic tangent of *d* 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(Decimal)

Returns the absolute value of the specified Decimal.

### Signature

```
public static Decimal abs(Decimal d)
```

### Parameters

***d***

Type: Decimal

### Return Value

Type: Decimal

## abs(Double)

Returns the absolute value of the specified Double.

### Signature

```
public static Double abs(Double d)
```

### Parameters

***d***

Type: Double

### Return Value

Type: Double

## abs(Integer)

Returns the absolute value of the specified Integer.

### Signature

```
public static Integer abs(Integer i)
```

### Parameters

***i***

Type: Integer

### Return Value

Type: Integer

### Example

```
Integer i = -42;
Integer i2 = math.abs(i);
system.assertEquals(i2, 42);
```

## abs(Long)

Returns the absolute value of the specified Long.

### Signature

```
public static Long abs(Long l)
```

### Parameters

***l***

Type: Long

#### Return Value

Type: Long

### acos(Decimal)

Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

#### Signature

```
public static Decimal acos(Decimal d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

### acos(Double)

Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

#### Signature

```
public static Double acos(Double d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### asin(Decimal)

Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.

#### Signature

```
public static Decimal asin(Decimal d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

## asin(Double)

Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.

### Signature

```
public static Double asin(Double d)
```

### Parameters

***d***

Type: Double

### Return Value

Type: Double

## atan(Decimal)

Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

### Signature

```
public static Decimal atan(Decimal d)
```

### Parameters

***d***

Type: Decimal

### Return Value

Type: Decimal

## atan(Double)

Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

### Signature

```
public static Double atan(Double d)
```

### Parameters

***d***

Type: Double

### Return Value

Type: Double

## atan2(Decimal, Decimal)

Converts rectangular coordinates (*x* and *y*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *x*/*y* in the range of -*pi* to *pi*.

#### Signature

```
public static Decimal atan2(Decimal x, Decimal y)
```

#### Parameters

***x***

Type: Decimal

***y***

Type: Decimal

#### Return Value

Type: Decimal

### atan2(Double, Double)

Converts rectangular coordinates (*x* and *y*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *x*/*y* in the range of *-pi* to *pi*.

#### Signature

```
public static Double atan2(Double x, Double y)
```

#### Parameters

***x***

Type: Double

***y***

Type: Double

#### Return Value

Type: Double

### cbrt(Decimal)

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 d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

## cbrt(Double)

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 d)
```

### Parameters

***d***

Type: Double

### Return Value

Type: Double

## ceil(Decimal)

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 d)
```

### Parameters

***d***

Type: Decimal

### Return Value

Type: Decimal

## ceil(Double)

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 d)
```

### Parameters

***d***

Type: Double

### Return Value

Type: Double

## cos(Decimal)

Returns the trigonometric cosine of the angle specified by *d*.

### Signature

```
public static Decimal cos(Decimal d)
```

### Parameters

***d***

Type: Decimal

### Return Value

Type: Decimal

## cos(Double)

Returns the trigonometric cosine of the angle specified by *d*.

### Signature

```
public static Double cos(Double d)
```

### Parameters

***d***

Type: Double

### Return Value

Type: Double

## cosh(Decimal)

Returns the hyperbolic cosine of *d*. 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 d)
```

### Parameters

***d***

Type: Decimal

### Return Value

Type: Decimal

## cosh(Double)

Returns the hyperbolic cosine of *d*. 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 d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### exp(Decimal)

Returns Euler's number *e* raised to the power of the specified Decimal.

#### Signature

```
public static Decimal exp(Decimal d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

### exp(Double)

Returns Euler's number *e* raised to the power of the specified Double.

#### Signature

```
public static Double exp(Double d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### floor(Decimal)

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 d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

### floor(Double)

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 d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### log(Decimal)

Returns the natural logarithm (base *e*) of the specified Decimal.

#### Signature

```
public static Decimal log(Decimal d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

### log(Double)

Returns the natural logarithm (base *e*) of the specified Double.

#### Signature

```
public static Double log(Double d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### log10(Decimal)

Returns the logarithm (base *10*) of the specified Decimal.

#### Signature

```
public static Decimal log10(Decimal d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

### log10(Double)

Returns the logarithm (base *10*) of the specified Double.

#### Signature

```
public static Double log10(Double d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### max(Decimal, Decimal)

Returns the larger of the two specified Decimals.

#### Signature

```
public static Decimal max(Decimal d1, Decimal d2)
```

#### Parameters

***d1***

Type: Decimal

***d2***

Type: Decimal

#### Return Value

Type: Decimal

#### Example

```
Decimal larger = math.max(12.3, 156.6);
system.assertEquals(larger, 156.6);
```

### max(Double, Double)

Returns the larger of the two specified Doubles.

#### Signature

```
public static Double max(Double d1, Double d2)
```

#### Parameters

***d1***

Type: Double

***d2***

Type: Double

#### Return Value

Type: Double

### max(Integer, Integer)

Returns the larger of the two specified Integers.

#### Signature

```
public static Integer max(Integer i1, Integer i2)
```

#### Parameters

***i1***

Type: Integer

***i2***

Type: Integer

#### Return Value

Type: Integer

### max(Long, Long)

Returns the larger of the two specified Longs.

#### Signature

```
public static Long max(Long l1, Long l2)
```

#### Parameters

***l1***

Type: Long

***l2***

Type: Long

#### Return Value

Type: Long

### min(Decimal, Decimal)

Returns the smaller of the two specified Decimals.

#### Signature

```
public static Decimal min(Decimal d1, Decimal d2)
```

#### Parameters

***d1***

Type: Decimal

***d2***

Type: Decimal

#### Return Value

Type: Decimal

#### Example

```
Decimal smaller = math.min(12.3, 156.6);
system.assertEquals(smaller, 12.3);
```

### min(Double, Double)

Returns the smaller of the two specified Doubles.

#### Signature

```
public static Double min(Double d1, Double d2)
```

#### Parameters

***d1***

Type: Double

***d2***

Type: Double

#### Return Value

Type: Double

## min(Integer, Integer)

Returns the smaller of the two specified Integers.

### Signature

```
public static Integer min(Integer i1, Integer i2)
```

### Parameters

***i1***

Type: Integer

***i2***

Type: Integer

### Return Value

Type: Integer

## min(Long, Long)

Returns the smaller of the two specified Longs.

### Signature

```
public static Long min(Long l1, Long l2)
```

### Parameters

***l1***

Type: Long

***l2***

Type: Long

### Return Value

Type: Long

## mod(Integer, Integer)

Returns the remainder of *i1* divided by *i2*.

### Signature

```
public static Integer mod(Integer i1, Integer i2)
```

### Parameters

***i1***

Type: Integer

***i2***

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(Long, Long)

Returns the remainder of *L1* divided by *L2*.

#### Signature

```
public static Long mod(Long L1, Long L2)
```

#### Parameters

***L1***

Type: Long

***L2***

Type: Long

#### Return Value

Type: Long

### pow(Double, Double)

Returns the value of the first Double raised to the power of *exp*.

#### Signature

```
public static Double pow(Double d, Double exp)
```

#### Parameters

***d***

Type: Double

***exp***

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(Decimal)

Returns the value that is closest in value to `d` and is equal to a mathematical integer.

#### Signature

```
public static Decimal rint(Decimal d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

### rint(Double)

Returns the value that is closest in value to `d` and is equal to a mathematical integer.

#### Signature

```
public static Double rint(Double d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### round(Double)

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 d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Integer

### round(Decimal)

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 d)
```

#### Parameters

***d***

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(Decimal)

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 d)
```

#### Parameters

***d***

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(Double)

Returns the closest Long to the specified Double.

### Signature

```
public static Long roundToLong(Double d)
```

### Parameters

***d***

Type: Double

### Return Value

Type: Long

## signum(Decimal)

Returns the signum function of the specified Decimal, which is 0 if *d* is 0, 1.0 if *d* is greater than 0, -1.0 if *d* is less than 0.

### Signature

```
public static Decimal signum(Decimal d)
```

### Parameters

***d***

Type: Decimal

### Return Value

Type: Decimal

## signum(Double)

Returns the signum function of the specified Double, which is 0 if *d* is 0, 1.0 if *d* is greater than 0, -1.0 if *d* is less than 0.

### Signature

```
public static Double signum(Double d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### sin(Decimal)

Returns the trigonometric sine of the angle specified by `d`.

#### Signature

```
public static Decimal sin(Decimal d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

### sin(Double)

Returns the trigonometric sine of the angle specified by `d`.

#### Signature

```
public static Double sin(Double d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### sinh(Decimal)

Returns the hyperbolic sine of `d`. The hyperbolic sine of `d` is defined to be $(e^x - e^{-x})/2$ where $e$ is Euler's number.

#### Signature

```
public static Decimal sinh(Decimal d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

### sinh(Double)

Returns the hyperbolic sine of `d`. The hyperbolic sine of `d` is defined to be $(e^x - e^{-x})/2$ where $e$ is Euler's number.

#### Signature

```
public static Double sinh(Double d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

### sqrt(Decimal)

Returns the correctly rounded positive square root of `d`.

#### Signature

```
public static Decimal sqrt(Decimal d)
```

#### Parameters

***d***

Type: Decimal

#### Return Value

Type: Decimal

### sqrt(Double)

Returns the correctly rounded positive square root of `d`.

#### Signature

```
public static Double sqrt(Double d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

## tan(Decimal)

Returns the trigonometric tangent of the angle specified by *d*.

### Signature

```
public static Decimal tan(Decimal d)
```

### Parameters

***d***

Type: Decimal

### Return Value

Type: Decimal

## tan(Double)

Returns the trigonometric tangent of the angle specified by *d*.

### Signature

```
public static Double tan(Double d)
```

### Parameters

***d***

Type: Double

### Return Value

Type: Double

## tanh(Decimal)

Returns the hyperbolic tangent of *d*. The hyperbolic tangent of *d* 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 d)
```

### Parameters

***d***

Type: Decimal

### Return Value

Type: Decimal

## tanh(Double)

Returns the hyperbolic tangent of *d*. The hyperbolic tangent of *d* 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 d)
```

#### Parameters

***d***

Type: Double

#### Return Value

Type: Double

# Pattern Class

Represents a compiled representation of a regular expression.

### Namespace

System

## Pattern Methods

The following are methods for `Pattern`.

***compile(String)***

Compiles the regular expression into a Pattern object.

***matcher(String)***

Creates a Matcher object that matches the input string *`regExp`* against this Pattern object.

***matches(String, String)***

Compiles the regular expression *`regExp`* and tries to match it against *`s`*. This method returns `true` if the string *`s`* matches the regular expression, `false` otherwise.

***pattern()***

Returns the regular expression from which this Pattern object was compiled.

***quote(String)***

Returns a string that can be used to create a pattern that matches the string *`s`* as if it were a literal pattern.

***split(String)***

Returns a list that contains each substring of the String that matches this pattern.

***split(String, Integer)***

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(String)

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(String)

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(String, String)

Compiles the regular expression *regExp* and tries to match it against *s*. This method returns `true` if the string *s* matches the regular expression, `false` otherwise.

#### Signature

```
public static Boolean matches(String regExp, String s)
```

#### Parameters

***regExp***

Type: String

***s***

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(String)

Returns a string that can be used to create a pattern that matches the string `s` as if it were a literal pattern.

#### Signature

```
public static String quote(String s)
```

#### Parameters

***s***

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(String)

Returns a list that contains each substring of the String that matches this pattern.

#### Signature

```
public String[] split(String s)
```

#### Parameters

***s***

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 *s* does not match the pattern, the resulting list has just one element containing the original String.

### split(String, Integer)

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[]

## QuickAction Class

Use Apex to request and process publisher 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 line items 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 line item. Otherwise, if the inserted line items don't originate from a quick action, the `WhereFrom__c` field is set to `'NoAction'`.

```
trigger MyTrigger on Line_Item__c (before insert) {
    for (Line_Item__c c : Trigger.new) {
        if (c.getQuickActionName() == QuickAction.CreateLineItem) {
            c.WhereFrom__c = 'GlobaActionl';
```

```
        } else if (c.getQuickActionName() ==
Schema.Invoice_Statement__c.QuickAction.CreateLineItem) {
            c.WhereFrom__c = 'InvoiceAction';
        } else if (c.getQuickActionName() == null) {
            c.WhereFrom__c = 'NoAction';
        } else {
            System.assert(false);
        }
    }
}
```

This sample performs a global action—`QuickAction.CreateLineItem`–on the passed-in line item object.

```
public Id globalCreate(Line_Item__c c) {
    QuickAction.QuickActionRequest req = new QuickAction.QuickActionRequest();
    req.quickActionName = QuickAction.CreateLineItem;
    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.

***describeAvailableQuickActions(String)***

Returns metadata information for the available quick actions of the provided parent object.

***describeQuickActions(List<String>)***

Returns the metadata information for the provided quick actions.

***performQuickAction(QuickAction.QuickActionRequest)***

Performs the quick action specified in the quick action request and returns the action result.

***performQuickAction(QuickAction.QuickActionRequest, Boolean)***

Performs the quick action specified in the quick action request with the option for partial success, and returns the result.

***performQuickActions(List<QuickAction.QuickActionRequest>)***

Performs the quick actions specified in the quick action request list and returns action results.

***performQuickActions(List<QuickAction.QuickActionRequest>, Boolean)***

Performs the quick actions specified in the quick action request list with the option for partial success, and returns action results.

### describeAvailableQuickActions(String)

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');
```

## describeQuickActions(List<String>)

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(QuickAction.QuickActionRequest)

Performs the quick action specified in the quick action request and returns the action result.

### Signature

```
public static QuickAction.QuickActionResult performQuickAction(QuickAction.QuickActionRequest
performQuickAction)
```

### Parameters

***performQuickAction***

Type: QuickAction.QuickActionRequest

### Return Value

Type: QuickAction.QuickActionResult

## performQuickAction(QuickAction.QuickActionRequest, Boolean)

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
performQuickAction, Boolean allOrNothing)
```

### Parameters

***performQuickAction***

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(List<QuickAction.QuickActionRequest>)

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> performQuickActions)
```

#### Parameters

***performQuickActions***

Type: List<QuickAction.QuickActionRequest>

#### Return Value

Type: List<QuickAction.QuickActionResult>

### performQuickActions(List<QuickAction.QuickActionRequest>, Boolean)

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> performQuickActions, Boolean
allOrNothing)
```

#### Parameters

***performQuickActions***

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>

## ResetPasswordResult Class

Represents the result of a password reset.

### Namespace

System

## ResetPasswordResult Methods

The following are instance methods for `ResetPasswordResult`.

***getPassword()***

Returns the password generated by the `System.resetPassword` method call.

### 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 Invoice_Statement__c doGet() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String invId = req.requestURI.substring(
                                req.requestURI.lastIndexOf('/')+1);
        Invoice_Statement__c result =
                       [SELECT Id, Description__c
                        FROM Invoice_Statement__c
                        WHERE Id = :invId];
        return result;
    }

}
```

## RestContext Properties

The following are properties for `RestContext`.

***request***

Returns the `RestRequest` for your Apex REST method.

***response***

Returns the `RestResponse` for your Apex REST method.

### 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='/Invoice_Statement__c/*')
global with sharing class MyRestResource {

    @HttpDelete
    global static void doDelete() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String invId = req.requestURI.substring(
                                req.requestURI.lastIndexOf('/')+1);
        Invoice_Statement__c inv =
                         [SELECT Id FROM Invoice_Statement__c
                          WHERE Id = :invId];
        delete inv;
    }

    @HttpGet
    global static Invoice_Statement__c doGet() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String invId = req.requestURI.substring(
                                 req.requestURI.lastIndexOf('/')+1);
        Invoice_Statement__c result =
```

```
                        [SELECT Id, Description__c
                         FROM Invoice_Statement__c
                         WHERE Id = :invId];
        return result;
    }

  @HttpPost
    global static String doPost(String status,
        String description) {
        Invoice_Statement__c inv = new Invoice_Statement__c();
        inv.Status__c = status;
        inv.Description__c = description;
        insert inv;
        return inv.Id;
    }
}
```



## RestRequest Constructors

The following are constructors for `RestRequest`.

***RestRequest()***

Creates a new instance of the `System.RestRequest` class.

### 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.

***headers***

Returns the headers that are received by the request.

***httpMethod***

Returns one of the supported HTTP request methods.

***params***

Returns the parameters that are received by the request.

***remoteAddress***

Returns the IP address of the client making the request.

***requestBody***

Returns or sets the body of the request.

***requestURI***

Returns or sets everything after the host in the HTTP request string.

***resourcePath***

Returns the REST resource path for the request.

## 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

## 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.



### addHeader(String, String)

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(String, String)

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 76.



## RestResponse Constructors

The following are constructors for `RestResponse`.

***RestResponse()***

Creates a new instance of the `System.RestResponse` class.

### 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.

***responseBody***

Returns or sets the body of the response.

***headers***

Returns the headers to be sent to the response.

***statusCode***

Returns or sets the response status code.

### 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 |
| 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 |

| Status Code | Description |
|---|---|
| 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.

### *addHeader(String, String)*

Adds a header to the response header map.

### addHeader(String, String)

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`.

***execute(SchedulableContext)***

Executes the scheduled Apex job.

### execute(SchedulableContext)

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`.

***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.

***getGlobalDescribe()***

Returns a map of all sObject names (keys) to sObject tokens (values) for the standard and custom objects defined in your organization.

***describeSObjects(List<String>)***

Describes metadata (field list and object properties) for the specified sObject or array of sObjects.

### 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();
```

### describeSObjects(List<String>)

Describes metadata (field list and object properties) for the specified sObject or array of sObjects.

#### Signature

```
public static List<Schema.DescribeSObjectResult> describeSObjects(List<String> types)
```

#### Parameters

***types***

Type: List<String>

The *types* 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 `describeGlobal` to retrieve a list of all objects for your organization, then iterate through the list and use `describeSObjects` to obtain metadata about individual objects. The `describeSObjects` method is limited to a maximum of 100 objects returned.

#### Example

```
Schema.DescribeSObjectResult[]
 descResult =
  Schema.describeSObjects(
   new String[]{
    'Merchandise__c',
    'Invoice_Statement__c'});
```

## Search Class

Used with dynamic SOSL queries.

### Namespace

System

## Search Methods

The following are static methods for `Search`.

***query(String)***

Creates a dynamic SOSL query at run time.

### query(String)

Creates a dynamic SOSL query at run time.

#### Signature

```
public static sObject[sObject[]] query(String query)
```

#### Parameters

***query***

Type: String

#### 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 SOQL.

## 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 27.

***Set Constructors***

***Set Methods***

## Set Constructors

The following are constructors for `Set`.

***Set<T>()***

Creates a new instance of the `Set` class. A set can hold elements of any data type T.

***Set<T>(Set<T>)***

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.

***Set<T>(List<T>)***

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.

### 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>(Set<T>)

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>(List<T>)

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.

***add(Object)***

Adds an element to the set if it is not already present.

***addAll(List<Object>)***

Adds all of the elements in the specified list to the set if they are not already present.

***addAll(Set<Object>)***

Adds all of the elements in the specified set to the set that calls the method if they are not already present.

***clear()***

Removes all of the elements from the set.

***clone()***

Makes a duplicate copy of the set.

***contains(Object)***

Returns `true` if the set contains the specified element.

***containsAll(List<Object>)***

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.

***containsAll(Set<Object>)***

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.

***equals(Set<Object>)***

Compares this set with the specified set and returns `true` if both sets are equal; otherwise, returns `false`.

***hashCode()***

Returns the hashcode corresponding to this set and its contents.

***isEmpty()***

Returns `true` if the set has zero elements.

***remove(Object)***

Removes the specified element from the set if it is present.

***removeAll(List<Object>)***

Removes the elements in the specified list from the set if they are present.

***removeAll(Set<Object>)***

Removes the elements in the specified set from the original set if they are present.

***retainAll(List<Object>)***

Retains only the elements in this set that are contained in the specified list.

***retainAll(Set)***

Retains only the elements in the original set that are contained in the specified set.

***size()***

Returns the number of elements in the set (its cardinality).

## add(Object)

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;
```

```
result = myString.add('d');
system.assertEquals(result, true);
```

## addAll(List<Object>)

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(Set<Object>)

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;
```

```
result1 = myString.addAll(sString);
system.assertEquals(result1, true);
```

## 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(Object)

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;
result = myString.contains('z');
system.assertEquals(result, false);
```

## containsAll(List<Object>)

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(Set<Object>)

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(result1, true);

result2 = myString.containsAll(rString);
system.assertEquals(result2, true);
```

## equals(Set<Object>)

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;
result = mySet.isEmpty();
system.assertEquals(result, true);
```

### remove(Object)

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(List<Object>)

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(result, true);

Integer result2 = mySet.size();
System.assertEquals(result2, 1);
```

### removeAll(Set<Object>)

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(List<Object>)

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(result, true);
```

### retainAll(Set)

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(result, true);

Integer result2 = mySet.size();
System.assertEquals(result2, 2);
```

## 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. The following are the instance methods for sObjects.

For more information on sObjects, see sObject Types on page 89.

## SObject Methods

The following are methods for `SObject`. All are instance methods.

### *addError(String)*

Marks a record with a custom error message and prevents any DML operation from occurring.

### *addError(Exception)*

Marks a record with a custom error message and prevents any DML operation from occurring.

### *field.addError(String)*

Places the specified error message on the field in the Database.com user interface and prevents any DML operation from occurring.

### *clear()*

Clears all field values

### *clone(Boolean, Boolean, Boolean, Boolean)*

Creates a copy of the sObject record.

### *get(String)*

Returns the value for the field specified by *`fieldName`*, such as .

### *get(Schema.sObjectField)*

Returns the value for the field specified by the field token `Schema.`***`sObjectField`***, such as, `Schema.Merchandise__c.Price__c`.

### *getOptions()*

Returns the database.DMLOptions object for the sObject.

### *getSObject(String)*

Returns the value for the specified field. This method is primarily used with dynamic DML to access values for external IDs.

### *getSObject(Schema.SObjectField)*

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(String)*

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(Schema.SObjectType)*

Returns the value for the field specified by the field token `Schema.`***`fieldName`***, such as, . 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 publisher action associated with this sObject. Typically used in triggers.

#### *put(String, Object)*

Sets the value for the specified field and returns the previous value for the field.

#### *put(Schema.SObjectField, Object)*

Sets the value for the field specified by the field token `Schema.sObjectField`, such as, `Schema.Merchandise__c.Price__c` and returns the previous value for the field.

#### *putSObject(String, sObject)*

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(Schema.sObjectType, sObject)*

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(Database.DMLOptions)*

Sets the DMLOptions object for the sObject.

## addError(String)

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.

## addError(Exception)

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.

### *field*.addError(String)

Places the specified error message on the field in the Database.com 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.
- 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.

#### Example

```
Trigger.new[0].myField__C.addError('bad');
```

### clear()

Clears all field values

### Signature

```
public Void clear()
```

### Return Value

Type: Void

## clone(Boolean, Boolean, Boolean, Boolean)

Creates a copy of the sObject record.

### Signature

```
public sObject clone(Boolean opt_preserve_id, Boolean opt_IsDeepClone, Boolean
opt_preserve_readonly_timestamps, Boolean opt_preserve_autonumber)
```

### Parameters

***opt_preserve_id***

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.

***opt_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`.

***opt_preserve_readonly_timestamps***

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.

***opt_preserve_autonumber***

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 Salesforce.comAPI version 22.0 or earlier, the default value for the *opt_preserve_id* argument is `true`, that is, the ID is preserved.

## get(String)

Returns the value for the field specified by *fieldName*, such as .

### Signature

```
public Object get(String fieldName)
```

### Parameters

***fieldName***

Type: String

### Return Value

Type: Object

### Usage

For more information, see Dynamic SOQL.

## get(Schema.sObjectField)

Returns the value for the field specified by the field token `Schema.`***`sObjectField`***, such as, `Schema.Merchandise__c.Price__c`.

### Signature

```
public Object get(Schema.sObjectField field)
```

### Parameters

***field***

Type: Schema.SObjectField

### Return Value

Type: Object

### Usage

For more information, see Dynamic SOQL.

## getOptions()

Returns the database.DMLOptions object for the sObject.

### Signature

```
public Database.DMLOptions getOptions()
```

### Return Value

Type: Database.DMLOptions

## getSObject(String)

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

## getSObject(Schema.SObjectField)

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

## getSObjects(String)

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.

## getSObjects(Schema.SObjectType)

Returns the value for the field specified by the field token `Schema.fieldName`, such as, . 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.

## getQuickActionName()

Retrieves the name of a publisher action associated with this sObject. Typically used in triggers.

### Signature

```
public String getQuickActionName()
```

### Return Value

Type: String

## put(String, Object)

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

### put(Schema.SObjectField, Object)

Sets the value for the field specified by the field token `Schema.`***`sObjectField`***, such as, `Schema.Merchandise__c.Price__c` 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

### putSObject(String, sObject)

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

### putSObject(Schema.sObjectType, sObject)

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(Database.DMLOptions)

Sets the DMLOptions object for the sObject.

#### Signature

```
public Void setOptions(database.DMLOptions DMLOptions)
```

#### Parameters

***DMLOptions***

Type: Database.DMLOptions

#### 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 22.

## String Methods

The following are methods for `String`.

### *abbreviate(Integer)*

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.

### *abbreviate(Integer, Integer)*

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.

### *capitalize()*

Returns the current String with the first letter changed to title case.

### *center(Integer)*

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.

### *center(Integer, String)*

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.

### *compareTo(String)*

Compares two strings lexicographically, based on the Unicode value of each character in the Strings.

### *contains(String)*

Returns `true` if and only if the String that called the method contains the specified sequence of characters in the *`compString`*.

### *containsAny(String)*

Returns `true` if the current String contains any of the characters in the specified String; otherwise, returns `false`.

### *containsIgnoreCase(String)*

Returns `true` if the current String contains the specified sequence of characters without regard to case; otherwise, returns `false`.

### *containsNone(String)*

Returns `true` if the current String doesn't contain the specified sequence of characters; otherwise, returns `false`.

### *containsOnly(String)*

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(String)*

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(String)*

Returns the difference between the current String and the specified String.

### *endsWith(String)*

Returns `true` if the String that called the method ends with the specified *`suffix`*.

### *endsWithIgnoreCase(String)*

Returns `true` if the current String ends with the specified suffix; otherwise, returns `false`.

### *equals(String)*

Returns `true` if the *`compString`* is not null and represents the same binary sequence of characters as the String that called the method. This method is true whenever the `compareTo` method returns 0. For example:

### *equalsIgnoreCase(String)*

Returns `true` if the *`compString`* is not null and represents the same sequence of characters as the String that called the method, ignoring case. For example:

### *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(String)*

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(String, List<String>)*

Treat the current string as a pattern that should be used for substitution in the same manner as `apex:outputText`.

### *fromCharArray(List<Integer>)*

Returns a String from the values of the list of integers.

### *getCommonPrefix(List<String>)*

Returns the initial sequence of characters as a String that is common to all the specified Strings.

### *getLevenshteinDistance(String)*

Returns the Levenshtein distance between the current String and the specified String.

### *getLevenshteinDistance(String, Integer)*

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(String)*

Returns the index of the first occurrence of the specified substring. If the substring does not occur, this method returns -1.

### *indexOf(String, Integer)*

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(String)*

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(String)*

Returns the zero-based index of the first occurrence of a character that is not in the specified substring. Otherwise, returns -1.

### *indexOfDifference(String)*

Returns the zero-based index of the character where the current String begins to differ from the specified String.

### *indexOfIgnoreCase(String)*

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(String, Integer)*

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(String)*

Returns `true` if the specified String is white space, empty (''), or null; otherwise, returns `false`.

### *isEmpty(String)*

Returns `true` if the specified String is empty ('') or null; otherwise, returns `false`.

### *isNotBlank(String)*

Returns `true` if the specified String is not whitespace, not empty (''), and not null; otherwise, returns `false`.

### *isNotEmpty(String)*

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; otherwise, returns `false`.

### *join(Object, String)*

Joins the elements of the specified iterable object, such as a List, into a single String separated by the specified separator.

### *lastIndexOf(String)*

Returns the index of the last occurrence of the specified substring. If the substring does not occur, this method returns -1.

### *lastIndexOf(String, Integer)*

Returns the index of the last occurrence of the specified substring, starting from the character at index 0 and ending at the specified index.

### *lastIndexOfIgnoreCase(String)*

Returns the index of the last occurrence of the specified substring regardless of case.

### *lastIndexOfIgnoreCase(String, Integer)*

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(Integer)*

Returns the leftmost characters of the current String of the specified length.

### *leftPad(Integer)*

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(Integer, Integer)*

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.

### *remove(String)*

Removes all occurrences of the specified substring and returns the String result.

### removeEnd(String)

Removes the specified substring only if it occurs at the end of the String.

### removeEndIgnoreCase(String)

Removes the specified substring only if it occurs at the end of the String using a case-insensitive match.

### removeStart(String)

Removes the specified substring only if it occurs at the beginning of the String.

### removeStartIgnoreCase(String)

Removes the specified substring only if it occurs at the beginning of the String using a case-insensitive match.

### repeat(Integer)

Returns the current String repeated the specified number of times.

### repeat(String, Integer)

Returns the current String repeated the specified number of times using the specified separator to separate the repeated Strings.

### replace(String, String)

Replaces each substring of a string that matches the literal target sequence `target` with the specified literal replacement sequence `replacement`.

### replaceAll(String, String)

Replaces each substring of a string that matches the regular expression `regExp` with the replacement sequence `replacement`.

### replaceFirst(String, String)

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(Integer)

Returns the rightmost characters of the current String of the specified length.

### rightPad(Integer)

Returns the current String padded with spaces on the right and of the specified length.

### split(String, Integer)

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(String)*

Returns `true` if the String that called the method begins with the specified *`prefix`*.

### *startsWithIgnoreCase(String)*

Returns `true` if the current String begins with the specified prefix regardless of the prefix case.

### *stripHtmlTags(String)*

Removes HTML markup from the input string and returns the plain text.

### *substring(Integer)*

Returns a new String that begins with the character at the specified zero-based *`startIndex`* and extends to the end of the String.

### *substring(Integer, Integer)*

Returns a new String that begins with the character at the specified zero-based *`startIndex`* and extends to the character at *`endIndex`* - 1.

### *substringAfter(String)*

Returns the substring that occurs after the first occurrence of the specified separator.

### *substringAfterLast(String)*

Returns the substring that occurs after the last occurrence of the specified separator.

### *substringBefore(String)*

Returns the substring that occurs before the first occurrence of the specified separator.

### *substringBeforeLast(String)*

Returns the substring that occurs before the last occurrence of the specified separator.

### *substringBetween(String)*

Returns the substring that occurs between two instances of the specified String.

### *substringBetween(String, String)*

Returns the substring that occurs between the two specified Strings.

### *swapCase(String, String)*

Swaps the case of all characters and returns the resulting String.

### *toLowerCase()*

Converts all of the characters in the String to lowercase using the rules of the default locale.

### *toLowerCase(String)*

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 locale.

### *toUpperCase(String)*

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(Date)*

Returns a String that represents the specified Date in the standard “yyyy-MM-dd” format.

### *valueOf(Datetime)*

Returns a String that represents the specified Datetime in the standard “yyyy-MM-dd HH:mm:ss” format for the local time zone.

### *valueOf(Decimal)*

Returns a String that represents the specified Decimal.

### *valueOf(Double)*

Returns a String that represents the specified Double.

### *valueOf(Integer)*

Returns a String that represents the specified Integer.

### *valueOf(Long)*

Returns a String that represents the specified Long.

### *valueOf(Object)*

Returns a string representation of the specified object argument.

### *valueOfGmt(Datetime)*

Returns a String that represents the specified Datetime in the standard “yyyy-MM-dd HH:mm:ss” format for the GMT time zone.

## abbreviate(Integer)

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(Integer, Integer)

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(Integer)

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(Integer, String)

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 padStr)
```

#### Parameters

***size***

Type: Integer

***padStr***

Type: String

#### Return Value

Type: String

#### Example

```
String s = 'hello';
String s2 =
   s.center(9);
System.assertEquals(
   '--hello--',
   s2);
```

## compareTo(String)

Compares two strings lexicographically, based on the Unicode value of each character in the Strings.

#### Signature

```
public Integer compareTo(String compString)
```

#### Parameters

***compString***

Type: String

#### Return Value

Type: Integer

#### Usage

The result is:

- A negative Integer if the String that called the method lexicographically precedes *compString*
- A positive Integer if the String that called the method lexicographically follows *compString*
- 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(String)

Returns `true` if and only if the String that called the method contains the specified sequence of characters in the *compString*.

#### Signature

```
public Boolean contains(String compString)
```

#### Parameters

***compString***

Type: String

#### Return Value

Type: Boolean

#### Example

```
String myString1 = 'abcde';
String myString2 = 'abcd';
Boolean result =
   myString1.contains(myString2);
System.assertEquals(result, true);
```

### containsAny(String)

Returns `true` if the current String contains any of the characters in the specified String; otherwise, returns `false`.

#### Signature

```
public Boolean containsAny(String compString)
```

#### Parameters

***compString***

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(String)

Returns `true` if the current String contains the specified sequence of characters without regard to case; otherwise, returns `false`.

#### Signature

```
public Boolean containsIgnoreCase(String compString)
```

#### Parameters

***compString***

Type: String

#### Return Value

Type: Boolean

#### Example

```
String s = 'hello';
Boolean b =
   s.containsIgnoreCase('HE');
System.assertEquals(
   true,
   b);
```

### containsNone(String)

Returns `true` if the current String doesn't contain the specified sequence of characters; otherwise, returns `false`.

#### Signature

```
public Boolean containsNone(String compString)
```

#### Parameters

***compString***

Type: String

If *compString* is an empty string or the current String is empty, this method returns `true`.If *compString* is null, this method returns a run-time exception.

#### Return Value

Type: Boolean

## containsOnly(String)

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 compString)
```

#### Parameters

***compString***
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

## countMatches(String)

Returns the number of times the specified substring occurs in the current String.

#### Signature

```
public Integer countMatches(String compString)
```

#### Parameters

***compString***

Type: String

#### Return Value

Type: Integer

### deleteWhitespace()

Returns a version of the current String with all white space characters removed.

#### Signature

```
public String deleteWhitespace()
```

#### Return Value

Type: String

### difference(String)

Returns the difference between the current String and the specified String.

#### Signature

```
public String difference(String compString)
```

#### Parameters

***compString***

Type: String

If *compString* is an empty string, this method returns an empty string.If *compString* 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(String)

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

## endsWithIgnoreCase(String)

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

## equals(String)

Returns `true` if the *`compString`* is not null and represents the same binary sequence of characters as the String that called the method. This method is true whenever the `compareTo` method returns 0. For example:

### Signature

```
public Boolean equals(String compString)
```

### Parameters

***`compString`***
Type: String

### Return Value

Type: Boolean

### Usage

Note that the == operator also performs String comparison, but is case-insensitive to match Apex semantics. (== is case-sensitive for ID comparison for the same reason.)

#### Example

```
String myString1 = 'abcde';
String myString2 = 'abcd';
Boolean result =
   myString1.equals(myString2);
System.assertEquals(result, false);
```

### equalsIgnoreCase(String)

Returns `true` if the *`compString`* is not null and represents the same sequence of characters as the String that called the method, ignoring case. For example:

#### Signature

```
public Boolean equalsIgnoreCase(String compString)
```

#### Parameters

***`compString`***

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(String)

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.

### 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(String, List<String>)

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

## fromCharArray(List<Integer>)

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

## getCommonPrefix(List<String>)

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(String)

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(String, Integer)

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(String)

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

## indexOf(String, Integer)

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 = 'abcd';
String myString2 = 'bc';
Integer result =
   myString1.indexOf(myString2, 0);
System.assertEquals(1, result);
```

## indexOfAny(String)

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(String)

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);
```

### indexOfDifference(String)

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(String)

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(String, Integer)

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

### isAllUpperCase()

Returns `true` if all characters in the current String are uppercase; otherwise, returns `false`.

#### Signature

```
public Boolean isAllUpperCase()
```

#### Return Value

Type: Boolean

### 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

## 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

## isAsciiPrintable()

Returns `true` if the current String contains only ASCII printable characters; otherwise, returns `false`.

### Signature

```
public Boolean isAsciiPrintable()
```

### Return Value

Type: Boolean

## isBlank(String)

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

## isEmpty(String)

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

## isNotBlank(String)

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

### isNotEmpty(String)

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

### 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.

### 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.

## isWhitespace()

Returns `true` if the current String contains only white space characters; otherwise, returns `false`.

### Signature

```
public Boolean isWhitespace()
```

### Return Value

Type: Boolean

## join(Object, String)

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(String)

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

### lastIndexOf(String, Integer)

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);
```

### lastIndexOfIgnoreCase(String)

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(String, Integer)

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(Integer)

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(Integer)

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(Integer, Integer)

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);
```

### remove(String)

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(String)

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(String)

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(String)

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(String)

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(Integer)

Returns the current String repeated the specified number of times.

#### Signature

```
public String repeat(Integer numTimes)
```

#### Parameters

***numTimes***

Type: Integer

#### Return Value

Type: String

#### Example

```
String s1 = 'SFDC';
String s2 =
   s1.repeat(2);
System.assertEquals(
   'SFDCSFDC', s2);
```

### repeat(String, Integer)

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 numTimes)
```

#### Parameters

***separator***

Type: String

***numTimes***

Type: Integer

#### Return Value

Type: String

#### Example

```
String s1 = 'SFDC';
String s2 =
   s1.repeat('-', 2);
System.assertEquals(
   'SFDC-SFDC', s2);
```

### replace(String, String)

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

## replaceAll(String, String)

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.

## replaceFirst(String, String)

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.

## reverse()

Returns a String with all the characters reversed.

### Signature

```
public String reverse()
```

#### Return Value

Type: String

### right(Integer)

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(Integer)

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(String, Integer)

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(String)

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

## startsWithIgnoreCase(String)

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

## stripHtmlTags(String)

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(Integer)

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

## substring(Integer, Integer)

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(String)

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(String)

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(String)

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(String)

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(String)

Returns the substring that occurs between two instances of the specified 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(String, String)

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(String, String)

Swaps the case of all characters and returns the resulting String.

#### Signature

```
public String swapCase(String open, String close)
```

#### Parameters

***open***

Type: String

***close***

Type: String

#### 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 locale.

#### Signature

```
public String toLowerCase()
```

#### Return Value

Type: String

### toLowerCase(String)

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

### toUpperCase()

Converts all of the characters in the String to uppercase using the rules of the default 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(String)

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

### 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.

### 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(Date)

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(Datetime)

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

## valueOf(Decimal)

Returns a String that represents the specified Decimal.

### Signature

```
public static String valueOf(Decimal decimalToConvert)
```

#### Parameters

***decimalToConvert***

Type: Decimal

#### Return Value

Type: String

### valueOf(Double)

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(Integer)

Returns a String that represents the specified Integer.

#### Signature

```
public static String valueOf(Integer integerToConvert)
```

#### Parameters

***integerToConvert***

Type: Integer

#### Return Value

Type: String

### valueOf(Long)

Returns a String that represents the specified Long.

#### Signature

```
public static String valueOf(Long longToConvert)
```

#### Parameters

***longToConvert***

Type: Long

#### Return Value

Type: String

### valueOf(Object)

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(Datetime)

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

# 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.

### *abortJob(String)*

Stops the specified job. The stopped job is still visible in the job queue in the Database.com user interface.

### *assert(Boolean, Object)*

Asserts that the specified condition is true. If it is not, a fatal error is returned that causes code execution to halt.

### *assertEquals(Object, Object, Object)*

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(Object, Object, Object)*

Asserts that the first two arguments are different. If they are the same, a fatal error is returned that causes code execution to halt.

### *currentTimeMillis()*

Returns the current time in milliseconds, which is expressed as the difference between the current time and midnight, January 1, 1970 UTC.

### *debug(Object)*

Writes the specified message, in string format, to the execution debug log. The `DEBUG` log level is used.

### *debug(LoggingLevel, Object)*

Writes the specified message, in string format, to the execution debug log with the specified log level.

### *getApplicationReadWriteMode()*

Returns the read write mode set for an organization during Salesforce.com upgrades and downtimes.

### *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(List<Ids>, String, String, String)*

Processes the list of work item IDs.

#### *purgeOldAsyncJobs(Date)*

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.

#### *resetPassword(ID, Boolean)*

Resets the password for the specified user.

#### *runAs(User)*

Changes the current user to the specified user.

#### *schedule(String, String, Object)*

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(Database.Batchable, String, Integer)*

Schedules a batch job to run once in the future after the specified time interval and with the specified job name.

#### *scheduleBatch(Database.Batchable, String, Integer, Integer)*

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(ID, String)*

Sets the password for the specified user.

#### *submit(List<ID>, String, String)*

Submits the processed approvals.

#### *today()*

Returns the current date in the current user's time zone.

## abortJob(String)

Stops the specified job. The stopped job is still visible in the job queue in the Database.com user interface.

### Signature

```
public static Void abortJob(String Job_ID)
```

### Parameters

***Job_ID***

Type: String

The *Job_ID* 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(Boolean, Object)

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 opt_msg)
```

### Parameters

***condition***

Type: Boolean

***opt_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(Object, Object, Object)

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 opt_msg)
```

### Parameters

***expected***

Type: Object

Specifies the expected value.

***actual***

Type: Object

Specifies the actual value.

***opt_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(Object, Object, Object)

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 opt_msg)
```

#### Parameters

***expected***

Type: Object

Specifies the expected value.

***actual***

Type: Object

Specifies the actual value.

***opt_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.

### 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(Object)

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 Database.com online help.

### debug(LoggingLevel, Object)

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 Database.com online help.

## 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`

#### Usage

`getApplicationReadWriteMode` is available as part of 5 Minute Upgrade.

#### 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 Database.com 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) {
        Invoice_Statement__c inv = new
            Invoice_Statement__c(
            Description__c='Invoice1');
        insert inv;
    }
  }
}
```

### 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(List<Ids>, String, String, String)

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(Date)

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.');
```

## resetPassword(ID, Boolean)

Resets the password for the specified user.

### Signature

```
public static System.ResetPasswordResult resetPassword(ID userID, Boolean send_user_email)
```

### Parameters

***userID***

Type: ID

***send_user_email***

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 *`send_user_email`*, the user is sent an email notifying them that their password was reset. A link to sign onto Database.com using the new password is included in the email. Use setPassword(ID, String) 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(User)

Changes the current user to the specified user.

#### Signature

```
public static Void runAs(User user_var)
```

#### Parameters

***`user_var`***
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 307.

**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(String, String, Object)

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 schedulable_class)
```

### Parameters

***JobName***

Type: String

***CronExpression***

Type: String

***schedulable_class***

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 100 that are allowed. 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:** Database.com 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 100 that are allowed. 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:** Database.com 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 |

| Name | Values | Special Characters |
|---|---|---|
| *Minutes* | 0–59 | None |
| *Hours* | 0–23 | , - * / |
| *Day_of_month* | 1–31 | , - * ? / L W |
| *Month* | 1–12 or the following:<br>• JAN<br>• FEB<br>• MAR<br>• APR<br>• MAY<br>• JUN<br>• JUL<br>• AUG<br>• SEP<br>• OCT<br>• NOV<br>• DEC | , - * / |
| *Day_of_week* | 1–7 or the following:<br>• SUN<br>• MON<br>• TUE<br>• WED<br>• THU<br>• FRI<br>• 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. |

| Special Character | Description |
|---|---|
| `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, 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.<br>  **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(Database.Batchable, String, Integer)

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`, Database.com 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.
- All scheduled Apex limits apply for batch jobs scheduled using `System.scheduleBatch`. After the batch job starts executing, 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(Database.Batchable, String, Integer, Integer)

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`, Database.com 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.
- All scheduled Apex limits apply for batch jobs scheduled using `System.scheduleBatch`. After the batch job starts executing, 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(ID, String)

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(ID, Boolean) 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(List<ID>, String, String)

Submits the processed approvals.

#### 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>

### 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.

***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.

***setFixedSearchResults(ID[])***

Defines a list of fixed search results to be returned by all subsequent SOSL statements in a test method.

#### *setMock(Type, Object)*

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(Boolean)*

Sets the application mode for an organization to read-only in an Apex test to simulate read-only mode during Database.com 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.

### 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

### setFixedSearchResults(ID[])

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[] opt_set_search_results)
```

#### Parameters

***opt_set_search_results***

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 309.

## setMock(Type, Object)

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

## setReadOnlyApplicationMode(Boolean)

Sets the application mode for an organization to read-only in an Apex test to simulate read-only mode during Database.com 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 application_mode)
```

### Parameters

***application_mode***

Type: Boolean

### Return Value

Type: Void

### Usage

`setReadOnlyApplicationMode` is available as part of 5 Minute Upgrade. See also the getApplicationReadWriteMode() System method.

### Example

The following example sets the application mode to read only and attempts to insert a new invoice statement 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 an invoice statement that is used for querying later.
    Invoice_Statement__c inv = new Invoice_Statement__c(
        Description__c='Test Invoice 1');
```

```
    insert inv;

    // 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 invoice statement object.
    Invoice_Statement__c inv2 = new Invoice_Statement__c(
        Description__c='Test Invoice 2');

    try {
      // Get the test invoice created earlier. Should be successful.
      Invoice_Statement__c testInvoiceFromDb =
        [SELECT Id FROM Invoice_Statement__c
         WHERE Description__c = 'Test Invoice 1'];
      System.assertEquals(inv.Id, testInvoiceFromDb.Id);

      // Inserts should result in the InvalidReadOnlyUserDmlException
      // being thrown.
      insert inv2;
      System.assertEquals(false, true);
    } catch (System.InvalidReadOnlyUserDmlException e) {
      // Expected
    }
    // Insertion should work after read only application mode gets disabled.
    Test.setReadOnlyApplicationMode(false);

    insert inv2;
    Invoice_Statement__c testInvoice2FromDb =
       [SELECT Id FROM Invoice_Statement__c
        WHERE Description__c = 'Test Invoice 2'];
    System.assertEquals(inv2.Id, testInvoice2FromDb.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.

## Time Class

Contains methods for the Time primitive data type.

### Namespace

System

### Usage

For more information on time, see Primitive Data Types on page 22.

## Time Methods

The following are methods for `Time`.

***addHours(Integer)***

Adds the specified number of hours to a Time.

***addMilliseconds(Integer)***

Adds the specified number of milliseconds to a Time.

***addMinutes(Integer)***

Adds the specified number of minutes to a Time.

***addSeconds(Integer)***

Adds the specified number of seconds to a Time.

***hour()***

Returns the hour component of a Time.

***millisecond()***

Returns the millisecond component of a Time.

***minute()***

Returns the minute component of a Time.

***newInstance(Integer, Integer, Integer, Integer)***

Constructs a Time from Integer representations of the specified hour, minutes, seconds, and milliseconds.

***second()***

Returns the second component of a Time.

## addHours(Integer)

Adds the specified number of hours to a Time.

### Signature

```
public Time addHours(Integer addlHours)
```

### Parameters

***addlHours***

Type: Integer

### Return Value

Type: Time

## addMilliseconds(Integer)

Adds the specified number of milliseconds to a Time.

### Signature

```
public Time addMilliseconds(Integer addlMilliseconds)
```

### Parameters

***addlMilliseconds***

Type: Integer

### Return Value

Type: Time

## addMinutes(Integer)

Adds the specified number of minutes to a Time.

### Signature

```
public Time addMinutes(Integer addlMinutes)
```

### Parameters

***addlMinutes***

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(Integer)

Adds the specified number of seconds to a Time.

#### Signature

```
public Time addSeconds(Integer addlSeconds)
```

#### Parameters

***addlSeconds***

Type: Integer

#### Return Value

Type: Time

## 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

### minute()

Returns the minute component of a Time.

#### Signature

```
public Integer minute()
```

#### Return Value

Type: Integer

### newInstance(Integer, Integer, Integer, Integer)

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

## 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
```

## TimeZone Methods

The following are methods for `TimeZone`.

#### *getDisplayName()*

Returns this time zone's display name.

#### *getID()*

Returns this time zone's ID.

***getOffset(Datetime)***

Returns the time zone offset, in milliseconds, of the specified date to the GMT time zone.

***getTimeZone(String)***

Returns the time zone corresponding to the specified time zone ID.

***toString()***

Returns the string representation of this time zone.

## 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(Datetime)

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(String)

Returns the time zone corresponding to the specified time zone ID.

#### Signature

```
public static TimeZone getTimeZone(String Id)
```

#### Parameters

***Id***

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

## 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 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.

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`.

***equals(Object)***

Returns `true` if the specified type is equal to the current type; otherwise, returns `false`.

***forName(String)***

Returns the type that corresponds to the specified fully qualified class name.

***forName(String, String)***

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(Object)

Returns `true` if the specified type is equal to the current type; otherwise, returns `false`.

#### Signature

```
public Boolean equals(Object toCompare)
```

#### Parameters

***toCompare***

Type: Object

The type to compare with the current type.

#### Return Value

Type: Boolean

#### Example

```
Type t1 = Merchandise__c.class;
Type t2 = Type.forName('Merchandise__c');
System.assert(t1.equals(t2));
```

## forName(String)

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

## forName(String, String)

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

### 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.

**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.com 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);
```

## URL Class

Represents a uniform resource locator (URL) and provides access to parts of the URL. Enables access to the base URL of a Database.com organization.

### Namespace

System

### Usage

Use the methods of the `System.URL` class to create links to objects in your organization. For example, you can create a link to a file uploaded as an attachment to a Chatter post by concatenating the Database.com 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 Database.com record. The full URL is created by concatenating the Database.com base URL with the record ID.

```
Invoice_Statement__c inv = [SELECT Id FROM Invoice_Statement__c
                            WHERE Description__c = 'My invoice' LIMIT 1];
String fullRecordURL = URL.getSalesforceBaseUrl().toExternalForm() + '/' + inv.Id;
```

### Example

In this example, the base URL and the full request URL for the current Database.com organization are retrieved. Next, a URL pointing to a specific invoice statement 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 invoice that we will create a link for later.
Invoice_Statement__c invoice = new Invoice_Statement__c(
                    Description__c='My invoice');
insert invoice;

// 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 invoice URL from the base URL.
String invoiceURL = URL.getSalesforceBaseUrl().toExternalForm() +
                        '/' + invoice.Id;
System.debug('URL of a particular invoice: ' + invoiceURL);

// 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());
```

***URL Constructors***
***URL Methods***

## URL Constructors

The following are constructors for `URL`.

#### *URL(String)*

Creates a new instance of the `URL` class using the specified string representation of the URL.

***URL(URL, String)***

Creates a new instance of the `URL` class by parsing the specified spec within the specified context.

***URL(String, String, String)***

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.

***URL(String, String, Integer, String)***

Creates a new instance of the `URL` class using the specified protocol, host, port, and file on the host.

## URL(String)

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(URL, String)

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 1033

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(String, String, String)

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(String, String, Integer, String)

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`.

***getAuthority()***

Returns the authority portion of the current URL.

***getCurrentRequestUrl()***

Returns the URL of an entire request for a Database.com organization.

***getDefaultPort()***

Returns the default port number of the protocol associated with the current URL.

***getFile()***

Returns the file name of the current URL.

***getFileFieldURL(String, String)***

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 base URL for a Database.com organization.

***getUserInfo()***

Gets the UserInfo portion of the current URL.

***sameFile(System.URL)***

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 for a Database.com organization.

#### Signature

```
public static System.URL getCurrentRequestUrl()
```

#### Return Value

Type: `System.URL`

#### Usage

### 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(String, String)

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 base URL for a Database.com organization.

#### Signature

```
public static System.URL getSalesforceBaseUrl()
```

#### Return Value

Type: `System.URL`

#### Usage

An example of an instance URL is `https://<unique_string>.database.com`. The unique string in the URL is unique for the organization.

### 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(System.URL)

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.

#### *getDefaultCurrency()*

Returns the context user's default currency code for multiple currency organizations or the organization's currency code for single currency organizations.

#### *getFirstName()*

Returns the context user's first name

#### *getLanguage()*

Returns the context user's language

#### *getLastName()*

Returns the context user's last name

#### *getLocale()*

Returns the context user's locale.

#### *getName()*

Returns the context user's full name. The format of the name depends on the language preferences specified for the organization.

#### *getOrganizationId()*

Returns the context organization's ID.

#### *getOrganizationName()*

Returns the context organization's company name.

#### *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.

***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 Salesforce.comAPI 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

### 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
}
```

***Version Constructors***
***Version Methods***

## Version Constructors

The following are constructors for `Version`.

***Version(Integer, Integer)***

Creates a new instance of the `Version` class as a two-part package version using the specified major and minor version numbers.

***Version(Integer, Integer, Integer)***

Creates a new instance of the `Version` class as a three-part package version using the specified major, minor, and patch version numbers.

## Version(Integer, Integer)

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(Integer, Integer, Integer)

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.

***compareTo(System.Version)***

Compares the current version with the specified version.

***major()***

Returns the major package version of the of the calling code.

***minor()***

Returns the minor package version of the calling code.

***patch()***

Returns the patch package version of the calling code or `null` if there is no patch version.

### compareTo(System.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

## 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 235.

## WebServiceMock Methods

The following are methods for `WebServiceMock`.

#### *doInvoke(Object, Object, Map<String, Object>, String, String, String, String, String, String)*

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.

### doInvoke(Object, Object, Map<String, Object>, String, String, String, String, String, String)

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 request, Map<String, Object> response, String
endpoint, String soapAction, String requestName, String responseNS, String responseName,
String responseType)
```

#### Parameters

***stub***

Type: Object

An instance of the auto-generated class.

***request***

Type: Object

The SOAP Web service request being invoked.

***response***

Type: Map<String, Object>

A collection of key/value pairs representing the response to send for the request.

When implementing this interface, set the *response* 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.

***responseNS***

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 `java.xml.stream.XMLStreamReader`.



## XmlStreamReader Constructors

The following are constructors for `XmlStreamReader`.

***XmlStreamReader(String)***

Creates a new instance of the `XmlStreamReader` class for the specified XML input.

### XmlStreamReader(String)

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.

***getAttributeCount()***

Returns the number of attributes on the start element, excluding namespace definitions.

***getAttributeLocalName(Integer)***

Returns the local name of the attribute at the specified index.

***getAttributeNamespace(Integer)***

Returns the namespace URI of the attribute at the specified index.

***getAttributePrefix(Integer)***

Returns the prefix of this attribute at the specified index.

***getAttributeType(Integer)***

Returns the XML type of the attribute at the specified index.

***getAttributeValue(String, String)***

Returns the value of the attribute in the specified *`localName`* at the specified URI.

***getAttributeValueAt(Integer)***

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(Integer)***

Returns the prefix for the namespace declared at the index.

***getNamespaceURI(String)***

Return the URI for the given prefix.

### *getNamespaceURIAt(Integer)*

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(Boolean)*

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(Boolean)*

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(Integer)

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(Integer)

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(Integer)

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(Integer)

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(String, String)

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(Integer)

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(Integer)

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(String)

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(Integer)

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 "Database.com For Dummies" >
     ]>
  <foo a=\"b\">Name &Title;</foo>';
```

  The `getText` method returns `Database.com 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(Boolean)

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(Boolean)

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 `https://stax-utils.dev.java.net/nonav/javadoc/api/javax/xml/stream/XMLStreamWriter.html`.



**See Also:**

*Http Class*
*HttpRequest Class*
*HttpResponse Class*

## XmlStreamWriter Constructors

The following are constructors for `XmlStreamWriter`.

***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.

***close()***

Closes this instance of an XmlStreamWriter and free any resources associated with it.

***getXmlString()***

Returns the XML written by the XmlStreamWriter instance.

***setDefaultNamespace(String)***

Binds the specified URI to the default namespace. This URI is bound in the scope of the current START_ELEMENT – END_ELEMENT pair.

***writeAttribute(String, String, String, String)***

Writes an attribute to the output stream.

***writeCData(String)***

Writes the specified CData to the output stream.

***writeCharacters(String)***

Writes the specified text to the output stream.

***writeComment(String)***

Writes the specified comment to the output stream.

***writeDefaultNamespace(String)***

Writes the specified namespace to the output stream.

***writeEmptyElement(String, String, String)***

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(String, String)***

Writes the specified namespace to the output stream.

***writeProcessingInstruction(String, String)***

Writes the specified processing instruction.

***writeStartDocument(String, String)***

Writes the XML Declaration using the specified XML encoding and version.

***writeStartElement(String, String, String)***

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(String)

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(String, String, String, String)

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(String)

Writes the specified CData to the output stream.

#### Signature

```
public Void writeCData(String data)
```

#### Parameters

***data***

Type: String

#### Return Value

Type: Void

### writeCharacters(String)

Writes the specified text to the output stream.

#### Signature

```
public Void writeCharacters(String text)
```

#### Parameters

***text***

Type: String

#### Return Value

Type: Void

### writeComment(String)

Writes the specified comment to the output stream.

#### Signature

```
public Void writeComment(String data)
```

#### Parameters

***data***

Type: String

#### Return Value

Type: Void

## writeDefaultNamespace(String)

Writes the specified namespace to the output stream.

### Signature

```
public Void writeDefaultNamespace(String namespaceURI)
```

### Parameters

***namespaceURI***

Type: String

### Return Value

Type: Void

## writeEmptyElement(String, String, String)

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(String, String)

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(String, String)

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(String, String)

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(String, String, String)

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

# 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 Database.com. Use the `webService` keyword to define these methods. For more information, see Considerations for Using the `WebService` Keyword on page 189.

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 30.0, use endpoint 30.0:

```
https://na1.salesforce.com/services/Soap/s/30.0
```

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.com 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

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**<br>reference<br>**Properties**<br>Create, Filter, Group, Sort<br>**Description**<br>The Apex class whose tests are to be executed.<br>This field can't be updated. |
| `ExtendedStatus` | **Type**<br>string<br>**Properties**<br>Filter, Nillable, Sort<br>**Description**<br>The pass rate of the test run.<br>For example: "(4/6)". This means that four out of a total of six tests passed.<br>If the class fails to execute, this field contains the cause of the failure. |
| `ParentJobId` | **Type**<br>reference<br>**Properties**<br>Filter, Group, Nillable, Sort<br>**Description**<br>Read-only. Points to the AsyncApexJob that represents the entire test run.<br>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. |

| Field Name | Description |
| --- | --- |
| `Status` | **Type**<br>picklist<br>**Properties**<br>Filter, Group, Restricted picklist, Sort, Update<br>**Description**<br>The status of the job. Valid values are:<br>• `Queued`<br>• `Preparing`<br>• `Processing`<br>• `Aborted`<br>• `Completed`<br>• `Failed` |

### 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**<br>reference<br>**Properties**<br>Filter, Group, Sort<br>**Description**<br>The Apex class whose test methods were executed. |

| Field Name | Details |
| --- | --- |
| `ApexLogId` | **Type**<br>reference<br>**Properties**<br>Filter, Group, Nillable, Sort<br>**Description**<br>Points to the ApexLog for this test method execution if debug logging is enabled; otherwise, `null`. |
| `AsyncApexJobId` | **Type**<br>reference<br>**Properties**<br>Filter, Group, Nillable, Sort<br>**Description**<br>Read-only. Points to the AsyncApexJob that represents the entire test run.<br>This field points to the same object as `ApexTestQueueItem.ParentJobId`. |
| `Message` | **Type**<br>string<br>**Properties**<br>Filter, Nillable, Sort<br>**Description**<br>The exception error message if a test failure occurs; otherwise, `null`. |
| `MethodName` | **Type**<br>string<br>**Properties**<br>Filter, Group, Nillable, Sort<br>**Description**<br>The test method name. |
| `Outcome` | **Type**<br>picklist<br>**Properties**<br>Filter, Group, Restricted picklist, Sort<br>**Description**<br>The result of the test method execution. Can be one of these values:<br>• Pass<br>• Fail |

| Field Name | Details |
| --- | --- |
| | • CompileFail |
| `QueueItemId` | **Type**<br>reference<br>**Properties**<br>Filter, Group, Nillable, Sort<br>**Description**<br>Points to the ApexTestQueueItem which is the class that this test method is part of. |
| `StackTrace` | **Type**<br>string<br>**Properties**<br>Filter, Nillable, Sort<br>**Description**<br>The Apex stack trace if the test failed; otherwise, `null`. |
| `TestTimestamp` | **Type**<br>dateTime<br>**Properties**<br>Filter, Sort<br>**Description**<br>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 test database organization) 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 Invoice_Statement__c (before insert){ " +
      "  for(Invoice_Statement__c a:Trigger.new){ " +
      "   a.Description__c = 't1_UPDATE';}" +
      "}";

    String testClassBody = "@isTest private class TestT1{" +
      "    public static testmethod void test1(){" +
      "      Invoice_Statement__c a  = new Invoice_Statement__c(" +
      "          Description__c='TEST');" +
      "      insert(a);" +
      "      a = [SELECT Id,Description__c FROM Invoice_Statement__c WHERE Id=:a.Id];" +
      "      System.assert(a.Description__c.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.
- 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. |

| Name | Type | Description |
|---|---|---|
| `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:<br>• CompileClassResult<br>• CompileTriggerResult<br>• DeleteApexResult<br>• 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. |
| `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. |

| Name | Type | Description |
|---|---|---|
| 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 a test database organization.

### Syntax

```
CompileClassResult[] = compileClasses(string[] classList);
```

### Usage

Use this call to compile Apex classes in a test database 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*.

### 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 a test database organization.

### Syntax

```
CompileTriggerResult[] = compileTriggers(string[] triggerList);
```

### Usage

Use this call to compile the specified Apex triggers in a test database 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.

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. |
| 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. |

| Name | Type | Description |
|---|---|---|
| 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

The `compileAndTest()` call contains a request, CompileAndTestRequest with information about the Apex to be compiled. The request also contains this object which specifies information about the Apex to be tested. You can specify the same or different classes to be tested as being 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 in a production organization, this request is ignored and all unit tests defined for your organization are run.

A CompileAndTestRequest 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. |

| Name | Type | Description |
|---|---|---|
| packages | string*[] | Do not use after version 10.0. For earlier, unsupported releases, the content of the package to be tested. |

**Note:** Packages are not supported in Database.com.

* Link goes to the *SOAP API Developer's Guide*.

## RunTestsResult

The call returns information about whether or not the compilation of the specified Apex was successful and if the unit tests completed successfully.

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. |
| successes | RunTestSuccess[] | An array of one or more RunTestSuccesses 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. |

## 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. |
| 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. |

* Link goes to the *SOAP API Developer's Guide*.

## 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. |
| 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:<br>• `NONE`<br>• `DEBUGONLY`<br>• `DB`<br>• `PROFILING`<br>• `CALLOUT`<br>• `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:<br>• `Db`<br>• `Workflow` |

| Element Name | Type | Description |
|---|---|---|
| | | • Validation<br>• Callout<br>• Apex_code<br>• Apex_profiling<br>• All |
| LogCategoryLevel | string | Specifies the amount of information returned in the debug log. Only the Apex_code LogCategory uses the log category levels.<br>Valid log levels are (listed from lowest to highest):<br>• ERROR<br>• WARN<br>• INFO<br>• DEBUG<br>• FINE<br>• FINER<br>• FINEST |

# 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 Database.com functionality blended with Apex. Many of the syntactic and semantic features of Apex, along with common idioms, are illustrated in this application.

**Note:** The shipping invoice example requires custom objects and fields that you must create first.

### 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 |

| Name | Type | Description |
|---|---|---|
| Weight | Number | The weight of the item, used to calculate shipping costs |
| 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:

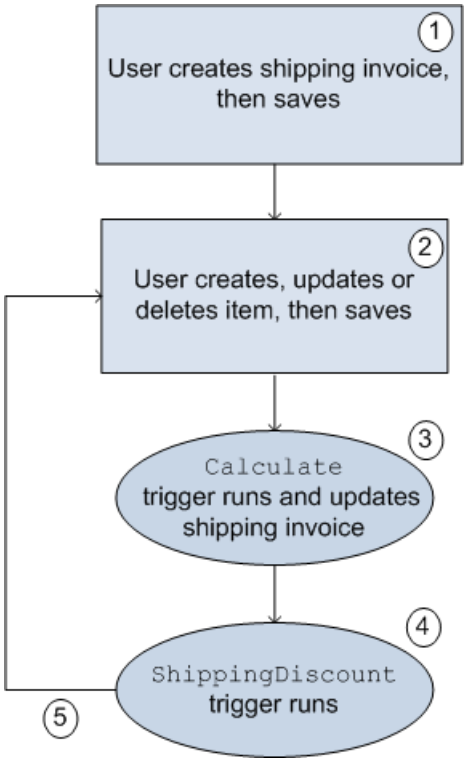



**Figure 8: Flow of user action and triggers for the shopping cart application**

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 2: 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* | super |
| bulk | last_90_days | switch* |
| by | last_month | synchronized* |
| byte* | last_n_days | system |
| case* | last_week | testmethod |
| cast* | like | then* |
| catch | limit | this |
| char* | list | this_month* |
| class | long | this_week |
| collect* | loop* | throw |
| commit | map | today |
| const* | merge | tolabel |
| continue | new | tomorrow |
| convertcurrency | next_90_days | transaction* |

| | | |
|---|---|---|
| decimal | next_month | trigger |
| default* | next_n_days | true |
| delete | next_week | try |
| desc | not | type* |
| do | null | undelete |
| else | nulls | update |
| end* | number* | upsert |
| enum | object* | using |
| exception | of* | virtual |
| exit* | on | webservice |
| export* | or | when* |
| extends | outer* | where |
| false | override | while |
| final | package | yesterday |
| finally | parallel* | |
| float* | pragma* | |
| for | private | |
| from | protected | |
| future | public | |
| global | | |
| goto* | | |
| group* | | |

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

## Documentation Typographical Conventions

Apex 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:<br>`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*];<br>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:<br>`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, anything included in brackets is optional. In the following example, specifying ***value*** is optional:<br>`data_type variable_name [ = value];` |
| \| | 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:<br>`Set<data_type> set_name`<br>`[= new Set<data_type>();] |`<br>`[= new Set<data_type>{value [, value2. . .] };] |`<br>`;` |

# 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 Database.com, 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 Database.com 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. Apex code can be initiated by Web service requests and from triggers on objects.

**Apex-Managed Sharing**

Enables developers to programmatically manipulate sharing to support their application's behavior. Apex-managed sharing is only available for custom objects.

**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.

**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.

**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 Database.com. 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, a line item has a child relationship with an invoice statement.

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 Database.com 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. See also Composite App and Native App.

**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.

**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, 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 Database.com 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.

**Database.com Certificate and Key Pair**

Database.com 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 Database.com certificate and key pair if you're working with an external website that wants verification that a request is coming from a Database.com organization.

**Data Loader**

A Force.com platform tool used to import and export data from your Database.com organization.

**Data Manipulation Language (DML)**

An Apex method or operation that inserts, updates, or deletes records from Database.com.

**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. Database.com 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**

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 Database.com 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.

**Developer Force**

The Developer Force website at developer.force.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 Database.com organization where you can make configuration changes that will not affect users on the production organization.For Database.com, the development environment is your test database organization.

## E

**Enterprise WSDL**

A strongly-typed WSDL for customers who want to build an integration with their Database.com 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 Database.com 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 Database.com 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

**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.

**Force.com**

The salesforce.com 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 Database.com 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 Database.com 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 Record ID.

**IdeaExchange**

A forum where salesforce.com customers can suggest new product concepts, promote favorite enhancements, interact with product managers and other customers, and preview what salesforce.com is planning to deliver in future releases. Visit IdeaExchange at ideas.salesforce.com.

**Instance**

The cluster of software and hardware represented as a single logical server that hosts an organization's data and runs their applications. Database.com runs on multiple instances, but data for any single organization is always consolidated 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 Database.com 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.

**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.

**Local Name**

The value stored for the field in the user'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 Database.com clears the lookup field. Alternatively, you can prevent records from being deleted if they're in a lookup relationship.

## M

**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, invoice statements have a master-detail relationship with line items. This type of relationship affects record deletion and security.

**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

**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-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 invoice statement may have one or more line items.

**Organization**

A deployment of Database.com with a defined set of licensed users. An organization is the virtual space provided to an individual customer of . 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.

**Owner**

Individual user to which a record is assigned.

## P

**PaaS**

See Platform as a Service.

**Partner WSDL**

A loosely-typed WSDL for customers, partners, and ISVs who want to build an integration or an app that can work across multiple Database.com 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 Database.com 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 Database.com edition based on Enterprise, Unlimited, or Performance Edition that does not include any of the standard Salesforce CRM 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 Database.com organization that has live users accessing data.

**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.

## R

**Record**

A single instance of a Database.com object.

**Record ID**

A unique 15- or 18-character alphanumeric string that identifies a single record in Database.com.

**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. Database.com 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 Database.com objects. A record name can be either free-form text or an autonumber field. `Record Name` does not have to be a unique value.

**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).

**Salesforce SOA (Service-Oriented Architecture)**

A powerful capability of Force.com that allows you to make calls to external Web services from within Apex.

**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. See also Anti-Join.

**Session ID**

An authentication token that is returned when a user successfully logs in to Database.com. The Session ID prevents a user from having to log in again every time he or she wants to perform another action in Database.com. Different from a record ID or Database.com ID, which are terms for the unique ID of a Database.com 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 Database.com 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.

**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.

**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 Database.com organization's information.

**sObject**

Any object that can be stored in Database.com.

**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.

**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**

A word or short phrases that can be associated with records to describe and organize their data in a personalized way. Administrators can enable tags for any custom objects. 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 Database**

A nearly identical copy of a Database.com production organization. You can create a test database for a variety of purposes, such as testing and training, without compromising the data and applications in your production environment.

**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**

A Database.com organization used strictly for testing. See also Test Database.

**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.

**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.

## 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 Database.com organization's information. See also SOAP API and Bulk API.

**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 Database.com include theAJAX Toolkit, which is a JavaScript wrapper around the Database.com SOAP API or wrapper classes created as part of a client integration application that accesses Database.com 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 Database.com Enterprise WSDL or Partner WSDL to communicate with Database.com 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



## M

## N



## O



## P



## Q

## R



## S

## T

## U



## V



## W



## X