
SOAP API Developer Guide

Version 55.0, Summer '22



CONTENTS

GET STARTED WITH SOAP API	1
Chapter 1: Introduction to SOAP API	1
Customize, Integrate, and Extend Your Salesforce Solutions	2
Supported Salesforce Editions	2
Standards Compliance	2
Development Platforms	2
SOAP API Support Policy	3
API End-of-Life	3
Choosing a WSDL	4
Related Resources	4
Quick Start: SOAP API	5
Step 1: Sign up for Salesforce Developer Edition	5
Step 2: Generate or Obtain the Web Service WSDL	5
Step 3: Import the WSDL File Into Your Development Platform	6
Step 4: Walk Through the Sample Code	7
Chapter 2: Core Data Types Used in API Calls	29
sObject	30
API Fault Element	30
ExceptionCode	31
Error	36
StatusCode	37
ExtendedErrorDetails	47
ExtendedErrorCode	47
Duplicate Management Data Types	47
DuplicateError	47
DuplicateResult	49
MatchResult	52
MatchRecord	53
FieldDiff	54
AdditionalInformationMap	55
Chapter 3: Tooling API Objects in the Enterprise WSDL	56
Chapter 4: API Call Basics	57
Characteristics of API Calls	58
Factors that Affect Data Access	58
Package Version Settings	60

Chapter 5: Error Handling	62
Error Handling for Session Expiration	63
More About Error Handling	63
Chapter 6: Security and the API	64
User Authentication	65
User Profile and Permission Sets Configuration	65
Security Token	65
Sharing	66
Implicit Restrictions for Objects and Fields	67
API Access in Salesforce AppExchange Packages	67
Outbound Port Restrictions	69
Chapter 7: Using the Partner WSDL	70
Obtaining the Partner WSDL File	71
Calls and the Partner WSDL	71
Objects, Fields, and Field Data and the Partner WSDL	72
Queries and the Partner WSDL	72
Namespaces in the Partner WSDL	73
Package Versions and the Partner WSDL	73
User Interface Themes	74
Examples Using the Partner WSDL	74
Sample query and queryMore Calls	78
Sample search Call	80
Sample create Call	83
Sample update Call	86
REFERENCE	89
Chapter 8: Apex-Related Calls	89
compileAndTest()	90
CompileAndTestRequest	91
CompileAndTestResult	92
compileClasses()	94
compileTriggers()	95
executeAnonymous()	95
ExecuteAnonymousResult	96
runTests()	96
RunTestsRequest	98
RunTestsResult	100
Chapter 9: Core Calls	105
convertLead()	106
LeadConvertResult	113

Contents

create()	113
SaveResult	123
delete()	123
DeleteResult	127
deleteByExample()	127
DeleteByExampleResult	129
emptyRecycleBin()	129
EmptyRecycleBinResult	132
executeListView()	132
ExecuteListViewRequest	133
ExecuteListViewResult	133
ListViewColumn	134
ListViewRecord	135
ListViewRecordColumn	135
findDuplicates()	135
findDuplicatesByIds()	139
getDeleted()	144
GetDeletedResult	148
getUpdated()	148
GetUpdatedResult	152
invalidateSessions()	152
InvalidateSessionsResult	154
login()	154
LoginResult	160
logout()	160
merge()	162
MergeResult	168
performQuickActions()	168
PerformQuickActionResult	170
process()	170
ProcessResult	173
query()	174
QueryResult	177
QueryLocator	180
queryAll()	180
queryMore()	183
QueryResult	187
QueryLocator	187
retrieve()	187
search()	190
SearchResult	194
undelete()	196
UndeleteResult	199
update()	199

Contents

SaveResult	206
upsert()	206
UpsertResult	211
Chapter 10: Describe Calls	213
describeAllTabs()	214
describeAppMenu()	215
DescribeAppMenuResult	216
describeApprovalLayout()	217
DescribeApprovalLayoutResult	219
describeAvailableQuickActions()	219
DescribeAvailableQuickActionResult	220
describeCompactLayouts()	221
DescribeCompactLayoutsResult	223
describeDataCategoryGroups()	224
DescribeDataCategoryGroupResult	226
describeDataCategoryGroupStructures()	227
describeDataCategoryGroupStructures()	231
describeGlobal()	231
DescribeGlobalResult	233
describeGlobalTheme()	235
DescribeGlobalTheme	237
describeKnowledge()	237
describeLayout()	238
DescribeLayoutResult	244
describePrimaryCompactLayouts()	256
describeQuickActions()	258
DescribeQuickActionResult	259
describeSearchScopeOrder()	264
DescribeSearchScopeOrderResult	265
describeSearchLayouts()	265
DescribeSearchLayoutResult	266
describeSObject()	267
describeSObjectResult	270
describeSObjects()	271
DescribeSObjectResult	275
describeSoftphoneLayout()	288
describeSoqlListView()	292
DescribeSoqlListView	293
DescribeSoqlListViewParams	293
DescribeSoqlListViewResult	294
DescribeSoqlListViewsRequest	294
ListViewColumn	294
ListViewOrderBy	295

Contents

SoqlWhereCondition	295
describeTabs()	297
describeTabSetResult	300
describeTheme()	302
DescribeThemeResult	304
DescribeThemeItem	304
Chapter 11: Utility Calls	305
changeOwnPassword()	305
getServerTimestamp()	307
getServerTimestampResult	309
getUserInfo()	309
getUserInfoResult	310
match()	312
MatchOptions	313
renderEmailTemplate()	314
RenderEmailTemplateResult	316
resetPassword()	318
sendEmail()	319
SendEmailResult	329
sendEmailMessage()	330
setPassword()	333
Chapter 12: SOAP Headers	335
AllOrNoneHeader	336
AllowFieldTruncationHeader	337
AssignmentRuleHeader	339
CallOptions	340
DisableFeedTrackingHeader	341
DebuggingHeader	342
DuplicateRuleHeader	344
EmailHeader	345
LimitInfoHeader	346
LocaleOptions	348
LoginScopeHeader	348
MruHeader	350
OwnerChangeOptions	351
PackageVersionHeader	354
QueryOptions	356
SessionHeader	356
UserTerritoryDeleteHeader	357
USING THE API WITH SALESFORCE FEATURES	358
Chapter 13: Implementation Considerations	358

Contents

Choosing a User for an Integration	359
Login Server URL	359
Log In to the Login Server	360
Typical API Call Sequence	360
Salesforce Sandbox	360
Multiple Instances of Salesforce Database Servers	360
Content Type Requirement	361
API Usage Metering	361
Compression	364
HTTP Persistent Connections	364
HTTP Chunking	365
Internationalization and Character Sets	365
XML Compliance	365
.NET, Non-String Fields, and the Enterprise WSDL	365
Chapter 14: Objects and SOAP API Calls and Headers for Apex	366
Chapter 15: Outbound Messaging	367
Understanding Outbound Messaging	368
Understanding Notifications	369
Setting Up Outbound Messaging	369
Setting Up User Profiles	370
Defining Outbound Messaging	370
Downloading the Salesforce Client Certificate	371
Viewing Outbound Messages	371
Tracking Outbound Message Status	372
Considerations for Security	372
Understanding the Outbound Messaging WSDL	372
Building a Listener	374
Chapter 16: Data Loading and Integration	376
Client Application Design	377
Salesforce Settings	377
Best Practices with Any Data Loader	378
Integration and Single Sign-On	379
Chapter 17: Data Replication	380
API Calls for Data Replication	381
Scope of Data Replication	381
Data Replication Steps	381
Object-Specific Requirements for Data Replication	382
Polling for Changes	382
Checking for Structural Changes in the Object	383
Chapter 18: Feature-Specific Considerations	384

Contents

Archived Activities	385
Person Account Record Types	385
External Objects	386
Call Centers and the API	387
Implementing Salesforce Integrations on Lightning Platform	389
Knowledge	389
GLOSSARY	394

GET STARTED WITH SOAP API

CHAPTER 1 Introduction to SOAP API

In this chapter ...

- [Customize, Integrate, and Extend Your Salesforce Solutions](#)
- [Supported Salesforce Editions](#)
- [Standards Compliance](#)
- [Development Platforms](#)
- [SOAP API Support Policy](#)
- [API End-of-Life](#)
- [Choosing a WSDL](#)
- [Related Resources](#)
- [Quick Start: SOAP API](#)

Salesforce provides programmatic access to your org's information using simple, powerful, and secure application programming interfaces. To use this document, you should have a basic familiarity with software development, web services, and the Salesforce user interface.

Any functionality described in this guide is available if your org has the API feature enabled. This feature is enabled by default for Performance, Unlimited, Enterprise, and Developer Editions. Some Professional Edition orgs have the API enabled. If you can't access the features you see in this guide, contact Salesforce.

Salesforce offers several APIs in addition to SOAP API. If you're wondering whether SOAP API is the right tool to use, check out [Which API Do I Use?](#) in *Salesforce Help*.



Note: Salesforce Education Services offers a suite of training courses to enable developers to design, create, integrate, and extend applications built on the Lightning platform. Be sure to visit <https://trailhead.salesforce.com/> to learn more.

Customize, Integrate, and Extend Your Salesforce Solutions

The Lightning Platform allows you to customize, integrate, and extend your Salesforce organization using the language and platform of your choice:

- **Customize Salesforce** with custom fields, links, objects, page layouts, buttons, record types, s-controls, and tabs to meet specific business requirements.
- **Integrate Salesforce** with your org's ERP and finance systems. Deliver real-time sales and support information to company portals and populate critical business systems with customer information.
- **Extend Salesforce** in presentation, business logic, and data services with new functionality that reflects the business requirements of your org.

For more information about Lightning Platform solutions and developer resources, go to [Salesforce Developers](#).

Supported Salesforce Editions

To use SOAP API, your org must use Enterprise Edition, Performance Edition, Unlimited Edition, or Developer Edition. If you're an existing Salesforce customer and want to upgrade to Enterprise, Unlimited, or Performance Edition, contact your account representative.

It is recommended that you use Developer Sandbox to develop Web service client applications. Developer Sandbox is an exact replica of your Salesforce deployment, including all customization and data. For more information, see [Deploy Enhancements from Sandboxes](#).

Developer Edition provides access to all features available with Enterprise Edition. Developer Edition is constrained only by the number of users and the amount of storage space. Developer Edition provides a development context that allows you to build and test your solutions without affecting your org's live data. Developer Edition accounts are available for free at

<https://developer.salesforce.com/gettingstarted>.


Standards Compliance

SOAP API is implemented to comply with the following specifications:

Standard Name	Website
Simple Object Access Protocol (SOAP) 1.1	https://www.w3.org/TR/2000/NOTE-SOAP-20000508/
Web Service Description Language (WSDL) 1.1	http://www.w3.org/TR/2001/NOTE-wsdl-20010315
WS-I Basic Profile 1.1	http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

Development Platforms

SOAP API works with current SOAP development environments, including, but not limited to, Visual Studio .NET 2005. In this document, we provide examples in Java and C# (.NET). The Java examples are based on WSC 20.0 (WSC) and JDK 6 (Java Platform Standard Edition Development Kit 6). Other versions of WSC are available at <https://github.com/forcedotcom/wsc> and <https://mvnrepository.com/artifact/com.force.api/force-wsc>. To see a complete list of compatible development platforms and more sample code, go to developer.salesforce.com.

 **Note:** Development platforms vary in their SOAP implementations. Implementation differences in certain development platforms might prevent access to some or all features of the API. If you are using Visual Studio for .NET development, we recommend that you use Visual Studio 2003 or higher.

SOAP API Support Policy

Salesforce recommends that your new client applications use the most recent version of the Lightning Platform WSDL file to fully exploit the benefits of richer features and greater efficiency. You can navigate to the most recent WSDL for your organization from Setup by entering *API* in the Quick Find box, then selecting **API**. When a new version is released, use the following steps in [Quick Start](#) to update your WSDL:

- Regenerate the WSDL file (see [Step 2: Generate or Obtain the Web Service WSDL](#))
- Import it into your environment (see [Step 3: Import the WSDL File Into Your Development Platform](#))

Backward Compatibility

Salesforce strives to make backward compatibility easy when using the Lightning Platform.

Each new Salesforce release consists of two components:

- A new release of platform software that resides on Salesforce systems
- A new version of SOAP API

For example, the Winter '07 release included SOAP API version 9.0 and the Summer '07 release included SOAP API version 10.0.

We maintain support for each SOAP API version across releases of the platform software. SOAP API is backward compatible in that an application created to work with a given SOAP API version will continue to work with that same SOAP API version in future platform software releases.

Salesforce does not guarantee that an application written against one SOAP API version will work with future SOAP API versions: Changes in method signatures and data representations are often required as we continue to enhance SOAP API. However, we strive to keep SOAP API consistent from version to version with minimal, if any, changes required to port applications to newer SOAP API versions.

For example, an application written using SOAP API version 9.0 which shipped with the Winter '07 release will continue to work with SOAP API version 9.0 on the Summer '07 release and on future releases beyond that. However, that same application may not work with SOAP API version 10 without modifications to the application.

API End-of-Life

Salesforce is committed to supporting each API version for a minimum of three years from the date of first release. In order to mature and improve the quality and performance of the API, versions that are more than three years old might cease to be supported.

When an API version is to be deprecated, advance notice is given at least one year before support ends. Salesforce will directly notify customers using API versions planned for deprecation.

Salesforce API Versions	Version Support Status	Version Retirement Info
Versions 31.0 through 55.0	Supported.	

Salesforce API Versions	Version Support Status	Version Retirement Info
Versions 21.0 through 30.0	As of Summer '22, these versions are deprecated and no longer supported by Salesforce. Starting from Summer '23, these versions will be retired and unavailable.	Salesforce Platform API Versions 21.0 through 30.0 Retirement
Versions 7.0 through 20.0	As of Summer '22, these versions are retired and unavailable.	Salesforce Platform API Versions 7.0 through 20.0 Retirement

If you request any resource or use an operation from a retired API version, SOAP API returns `500 : UNSUPPORTED_API_VERSION` error code.

To identify requests made from old or unsupported API versions of SOAP API, access the free [API Total Usage](#) event type.

Choosing a WSDL

There are two Lightning Platform Web services for which you can obtain WSDL files for API access:

- **Lightning Platform Enterprise WSDL**—This API is for most enterprise users who are developing client applications for their org. The enterprise WSDL file is a strongly typed representation of your org's data. It provides information about your schema, data types, and fields to your development environment, allowing for a tighter integration between it and the Lightning Platform Web service. This WSDL changes if custom fields or custom objects are added to, renamed, or removed from, your org's Salesforce configuration. If you are downloading an enterprise WSDL and you have managed packages installed in your organization, you need to take an extra step to select the version of each installed package to include in the generated WSDL.

Note the following when generating the enterprise WSDL:

- If new custom fields or objects are added to, renamed, or removed from your org's information, you must regenerate the WSDL file to access them.
 - The generated WSDL contains the objects and fields in your org, including those available in the selected versions of each installed package. If a field or object is added in a later package version, you must generate the enterprise WSDL with that package version to work with the object or field in your API integration.
- **Lightning Platform Partner WSDL**—This API is for Salesforce partners who are developing client applications for multiple orgs. As a loosely-typed representation of the Salesforce object model, the [partner WSDL](#) can be used to access data within any org.

Related Resources

The Salesforce developer website provides a full suite of developer toolkits, sample code, sample SOAP messages, community-based support, and other resources to help you with your development projects. Be sure to visit developer.salesforce.com for more information, or visit developer.salesforce.com/signup to sign up for a free Developer Edition account.

You can visit these websites to find out more about Salesforce applications:

- [Salesforce](#) for information about the Salesforce application.
- [Salesforce AppExchange](#) for access to apps created for Salesforce.
- [Trailblazer Community](#) for services to ensure Salesforce customer success.

Quick Start: SOAP API

Use this quick start to create a sample application in your development environment.

 **Note:** Before you begin building an integration or other client application:

- Install your development platform according to its product documentation.
- Read through all the steps before beginning this quick start. You may also wish to review the rest of this document to familiarize yourself with terms and concepts.

Step 1: Sign up for Salesforce Developer Edition

Use Salesforce Developer Edition to develop, stage, and test your API code against sample data.

Using a separate org to develop your applications protects your live data during testing. This recommendation is especially true for applications that insert, update, or delete data (as opposed to simply reading data). After you've tested your code, you can implement it in an edition with API access.

To create a Developer Edition org, go to developer.salesforce.com/signup and follow the instructions for signing up for a Developer Edition organization.

If you already have a Developer Edition organization, verify that your user profile has the API Enabled permission. This permission is enabled by default, but may have been changed by an administrator. For more information, see Salesforce Help.

Step 2: Generate or Obtain the Web Service WSDL

To access the Lightning Platform Web service, you need a Web Service Description Language (WSDL) file. The WSDL file defines the Web service that is available to you. Your development platform uses this WSDL to generate an API to access the Lightning Platform Web service it defines. You can either obtain the WSDL file from your organization's Salesforce administrator or you can generate it yourself if you have access to the WSDL download page in the Salesforce user interface. You can navigate to the most recent WSDL for your organization from Setup by entering `API` in the Quick Find box, then selecting **API**.

For more information about WSDL, see <http://www.w3.org/TR/wsdl>.

Generating the WSDL File for Your Organization

Any user with the Modify All Data permission can download the Web Services Description Language (WSDL) file to integrate and extend Salesforce using the API. (The System Administrator profile has this permission.)

The WSDL file is dynamically generated based on which type of WSDL file (enterprise or partner) you download. The generated WSDL defines all of the API calls, objects (including standard and custom objects), and fields that are available for API access for your organization.


To generate the WSDL file for your organization:

1. Log in to your Enterprise, Unlimited, Performance, or Developer Edition Salesforce account. You must log in as an administrator or as a user who has the "Modify All Data" permission. Logins are checked to ensure they are from a known IP address. For more information, see [Security and the API](#).
2. From Setup, enter `API` in the Quick Find box, then select **API** to display the WSDL download page.
3. Download the [appropriate WSDL](#):
 - If you're downloading an enterprise WSDL and you have managed packages installed in your org, click **Generate Enterprise WSDL**. Select the version of each installed package to include in the generated WSDL. By default, it is set to the latest installed versions of the packages.

- Otherwise, right-click the link for the appropriate WSDL document to save it to a local directory. In the menu, Internet Explorer users can choose **Save Target As**, while Google Chrome and Mozilla Firefox users can choose **Save Link As**.

Step 3: Import the WSDL File Into Your Development Platform

Once you have the WSDL file, you need to import it into your development platform so that your development environment can generate the necessary objects for use in building client Web service applications in that environment. This section provides sample instructions for WSC and Microsoft Visual Studio. For instructions about other development platforms, see your platform's product documentation.

 **Note:** The process for importing WSDL files is identical for the enterprise and partner WSDL files.

Instructions for Java Environments (WSC)

Java environments access the API through Java objects that serve as proxies for their server-side counterparts. Before using the API, you must first generate these objects from your organization's WSDL file.

Each SOAP client has its own tool for this process. For WSC, use the `wsd1c` utility.

 **Note:** Before you run `wsd1c`, you must have the WSC JAR file installed on your system and referenced in your classpath.

The basic syntax for `wsd1c` is:

```
java -classpath pathToJAR/wsc-22.jar com.sforce.ws.tools.wsd1c pathToWsd1/Wsd1Filename
pathToOutputJar/OutputJarFilename
```


This command generates an output jar file based on the specified WSDL file. After the output jar file is created, reference it along with the wsc jar file (for example, wsc-22.jar) in your Java program to create a client application.

Instructions for Microsoft Visual Studio

Visual Studio languages access the API through objects that serve as proxies for their server-side counterparts. Before using the API, you must first generate these objects from your organization's WSDL file.

Once you have the proxy classes for the server-side objects, you need to ensure that you specify whether you have set any values on non-string fields. For more information, see [Implementation Considerations](#).

Visual Studio provides two approaches for importing your WSDL file and generating an XML Web service client: an IDE-based approach and a command line approach. This walkthrough describes how to import your WSDL file through the IDE.

 **Note:** Before you begin, the first step is to create a new application or open an existing application in Visual Studio. In addition, you need to have generated the WSDL file, as described in [Generating the WSDL File for Your Organization](#).

An XML Web service client is any component or application that references and uses an XML Web service. This does not necessarily need to be a client-based application. In fact, in many cases, your XML Web service clients might be other Web applications, such as Web Forms or even other XML Web services. When accessing XML Web services in managed code, a proxy class and the .NET Framework handle all of the infrastructure coding.


To access an XML Web service from managed code:

1. Name your project `walkthrough` or change the `using` directive in the following sample to `your_project_name.web_reference_name`. Then, add a Web reference to your project for the XML Web service that you want to access. The Web reference creates a proxy class with methods that serve as proxies for each exposed method of the XML Web service.
2. Add the namespace for the Web reference.

3. Create an instance of the proxy class and then access the methods of that class as you would the methods of any other class.

You can add either a .NET 2.0 style Web reference, or a .NET 3.0 style Service reference, depending on your version of Visual Studio and preferred developer environment. A .NET 3.0 style reference uses services like SoapClient instead of SforceService.


To add a Web reference:

 **Note:** These steps may be different depending on the version of Visual Studio that you're using. For more information, see "Adding and Removing Web References" in the Visual Studio documentation.

1. If you are using Visual Studio 2010 or earlier, on the Project menu, choose **Add Web Reference**. For later versions of Visual Studio, on the Project menu, choose **Add Service Reference**, select **Advanced** and then select **Add Web Reference**.
2. In the URL box of the Add Web Reference dialog box, type the URL to obtain the service description of the XML Web service you want to access, such as:

```
c:\WSDLFiles\enterprise.wsdl
```

3. Click **Go** to retrieve information about the XML Web service.
4. In the Web reference name box, rename the Web reference to `sforce`, which is the name you will use for this Web reference.
5. Click **Add Reference** to add a Web reference for the target XML Web service.
6. Visual Studio retrieves the service description and generates a proxy class to interface between your application and the XML Web service.

 **Note:** If you are using Visual Basic .Net 1.1 and the enterprise WSDL, you will need to modify the generated Web service client to overcome a bug in Visual Studio's client generation utility. The API exposes two objects (`Case` and `Event`) whose names conflict with Visual Basic keywords. When the classes that represent these objects are created, Visual Studio wraps the class names with brackets (`[Case]` and `[Event]`). This is the method by which you can reuse keywords.

Unfortunately, in the definition of the `SObject` class, Visual Studio does not wrap `Case` and `Event` to class references in the `System.Xml.Serialization.XmlIncludeAttribute` that are part of the `SObject` definition. To work around this problem in Visual Studio, you need to edit the `XmlIncludeAttribute` settings for `Case` and `Event` as shown below. This does not apply to C# and only applies when using the enterprise version of the WSDL.

```
System.Xml.Serialization.XmlIncludeAttribute(GetType([Event])), _
System.Xml.Serialization.XmlIncludeAttribute(GetType([Case])), _
```

Step 4: Walk Through the Sample Code

Once you have imported your WSDL file, you can begin building client applications that use the API. Use the following samples to create a basic client application. Comments embedded in the sample explain each section of code.

Java Sample Code

This section walks through a sample Java client application that uses the WSC SOAP client. The purpose of this sample application is to show the required steps for logging into the login server and to demonstrate the invocation and subsequent handling of several API calls.

To run this sample, you must pass the authentication endpoint URL as an argument for your program. You can obtain this URL from the WSDL file. This sample application performs the following main tasks:

1. Prompts the user for their Salesforce username and password.

2. Calls `login()` to log in to the single login server and, if the login succeeds, retrieves user information and writes it to the console along with session information.
3. Calls `describeGlobal()` to retrieve a list of all available objects for the organization's data. The `describeGlobal` method determines [the objects that are available to the logged in user](#). This call should not be made more than once per session, since the data returned from the call is not likely to change frequently. The `DescribeGlobalResult` is echoed to the console.
4. Calls `describeSObjects()` to retrieve metadata (field list and object properties) for a specified object. The `describeSObject` method illustrates the type of metadata information that can be obtained for each object available to the user. The sample client application executes a `describeSObjects()` call on the object that the user specifies and then echoes the returned metadata information to the console. Object metadata information includes permissions, field types and lengths, and available values for picklist fields and types for `referenceTo` fields.
5. Calls `query()`, passing a simple query string (`"SELECT FirstName, LastName FROM Contact"`), and iterating through the returned `QueryResult`.
6. Calls `logout()` to log the user out.

The following sample code uses try/catch blocks to handle exceptions that might be thrown by the API calls.

```
package com.example.samples;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.InputStreamReader;
import java.io.IOException;
import com.sforce.soap.enterprise.DeleteResult;
import com.sforce.soap.enterprise.DescribeGlobalResult;
import com.sforce.soap.enterprise.DescribeGlobalSObjectResult;
import com.sforce.soap.enterprise.DescribeSObjectResult;
import com.sforce.soap.enterprise.EnterpriseConnection;
import com.sforce.soap.enterprise.Error;
import com.sforce.soap.enterprise.Field;
import com.sforce.soap.enterprise.FieldType;
import com.sforce.soap.enterprise.GetUserInfoResult;
import com.sforce.soap.enterprise.LoginResult;
import com.sforce.soap.enterprise.PicklistEntry;
import com.sforce.soap.enterprise.QueryResult;
import com.sforce.soap.enterprise.SaveResult;
import com.sforce.soap.enterprise.sobject.Account;
import com.sforce.soap.enterprise.sobject.Contact;
import com.sforce.soap.enterprise.sobject.SObject;
import com.sforce.ws.ConnectorConfig;
import com.sforce.ws.ConnectionException;

public class QuickstartApiSample {

    private static BufferedReader reader = new BufferedReader(
        new InputStreamReader(System.in));

    EnterpriseConnection connection;
    String authEndPoint = "";

    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Usage: com.example.samples."
```

```
        + "QuickstartApiSamples <AuthEndPoint>");

        System.exit(-1);
    }

    QuickstartApiSample sample = new QuickstartApiSample(args[0]);
    sample.run();
}

public void run() {
    // Make a login call
    if (login()) {
        // Do a describe global
        describeGlobalSample();

        // Describe an object
        describeSObjectsSample();

        // Retrieve some data using a query
        querySample();

        // Log out
        logout();
    }
}

// Constructor
public QuickstartApiSample(String authEndPoint) {
    this.authEndPoint = authEndPoint;
}

private String getUserInput(String prompt) {
    String result = "";
    try {
        System.out.print(prompt);
        result = reader.readLine();
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }

    return result;
}

private boolean login() {
    boolean success = false;
    String username = getUserInput("Enter username: ");
    String password = getUserInput("Enter password: ");

    try {
        ConnectorConfig config = new ConnectorConfig();
        config.setUsername(username);
        config.setPassword(password);

        System.out.println("AuthEndPoint: " + authEndPoint);
    }
}
```

```
        config.setAuthEndpoint(authEndPoint);

        connection = new EnterpriseConnection(config);
        printUserInfo(config);

        success = true;
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }

    return success;
}

private void printUserInfo(ConnectorConfig config) {
    try {
        GetUserInfoResult userInfo = connection.getUserInfo();

        System.out.println("\nLogging in ...\n");
        System.out.println("UserID: " + userInfo.getUserId());
        System.out.println("User Full Name: " + userInfo.getUserFullName());
        System.out.println("User Email: " + userInfo.getUserEmail());
        System.out.println();
        System.out.println("SessionID: " + config.getSessionId());
        System.out.println("Auth End Point: " + config.getAuthEndpoint());
        System.out
            .println("Service End Point: " + config.getServiceEndpoint());
        System.out.println();
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}

private void logout() {
    try {
        connection.logout();
        System.out.println("Logged out.");
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}

/**
 * To determine the objects that are available to the logged-in user, the
 * sample client application executes a describeGlobal call, which returns
 * all of the objects that are visible to the logged-in user. This call
 * should not be made more than once per session, as the data returned from
 * the call likely does not change frequently. The DescribeGlobalResult is
 * simply echoed to the console.
 */
private void describeGlobalSample() {
    try {
        // describeGlobal() returns an array of object results that
        // includes the object names that are available to the logged-in user.
        DescribeGlobalResult dgr = connection.describeGlobal();
    }
}
```

```

        System.out.println("\nDescribe Global Results:\n");
        // Loop through the array echoing the object names to the console
        for (int i = 0; i < dgr.getSobjects().length; i++) {
            System.out.println(dgr.getSobjects()[i].getName());
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}

/**
 * The following method illustrates the type of metadata information that can
 * be obtained for each object available to the user. The sample client
 * application executes a describeSObject call on a given object and then
 * echoes the returned metadata information to the console. Object metadata
 * information includes permissions, field types and length and available
 * values for picklist fields and types for referenceTo fields.
 */
private void describeSObjectsSample() {
    String objectToDescribe = getUserInput("\nType the name of the object to "
        + "describe (try Account): ");

    try {
        // Call describeSObjects() passing in an array with one object type
        // name
        DescribeSObjectResult[] dsrArray = connection
            .describeSObjects(new String[] { objectToDescribe });

        // Since we described only one sObject, we should have only
        // one element in the DescribeSObjectResult array.
        DescribeSObjectResult dsr = dsrArray[0];

        // First, get some object properties
        System.out.println("\n\nObject Name: " + dsr.getName());

        if (dsr.getCustom())
            System.out.println("Custom Object");
        if (dsr.getLabel() != null)
            System.out.println("Label: " + dsr.getLabel());

        // Get the permissions on the object

        if (dsr.getCreateable())
            System.out.println("Createable");
        if (dsr.getDeletable())
            System.out.println("Deletable");
        if (dsr.getQueryable())
            System.out.println("Queryable");
        if (dsr.getReplicateable())
            System.out.println("Replicateable");
        if (dsr.getRetrieveable())
            System.out.println("Retrieveable");
        if (dsr.getSearchable())

```

```
        System.out.println("Searchable");
    if (dsr.getUndeleteable())
        System.out.println("Undeleteable");
    if (dsr.getUpdateable())
        System.out.println("Updateable");

    System.out.println("Number of fields: " + dsr.getFields().length);

    // Now, retrieve metadata for each field
    for (int i = 0; i < dsr.getFields().length; i++) {
        // Get the field
        Field field = dsr.getFields()[i];

        // Write some field properties
        System.out.println("Field name: " + field.getName());
        System.out.println("\tField Label: " + field.getLabel());

        // This next property indicates that this
        // field is searched when using
        // the name search group in SOSL
        if (field.getNameField())
            System.out.println("\tThis is a name field.");

        if (field.getRestrictedPicklist())
            System.out.println("This is a RESTRICTED picklist field.");

        System.out.println("\tType is: " + field.getType());

        if (field.getLength() > 0)
            System.out.println("\tLength: " + field.getLength());

        if (field.getScale() > 0)
            System.out.println("\tScale: " + field.getScale());

        if (field.getPrecision() > 0)
            System.out.println("\tPrecision: " + field.getPrecision());

        if (field.getDigits() > 0)
            System.out.println("\tDigits: " + field.getDigits());

        if (field.getCustom())
            System.out.println("\tThis is a custom field.");

        // Write the permissions of this field
        if (field.getNillable())
            System.out.println("\tCan be nulled.");
        if (field.getCreateable())
            System.out.println("\tCreateable");
        if (field.getFilterable())
            System.out.println("\tFilterable");
        if (field.getUpdateable())
            System.out.println("\tUpdateable");

        // If this is a picklist field, show the picklist values
```

```

        if (field.getType().equals(FieldType.picklist)) {
            System.out.println("\t\tPicklist values: ");
            PicklistEntry[] picklistValues = field.getPicklistValues();

            for (int j = 0; j < field.getPicklistValues().length; j++) {
                System.out.println("\t\tValue: "
                    + picklistValues[j].getValue());
            }
        }

        // If this is a foreign key field (reference),
        // show the values
        if (field.getType().equals(FieldType.reference)) {
            System.out.println("\t\tCan reference these objects:");
            for (int j = 0; j < field.getReferenceTo().length; j++) {
                System.out.println("\t\t" + field.getReferenceTo()[j]);
            }
        }
        System.out.println("");
    }
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

private void querySample() {
    String soqlQuery = "SELECT FirstName, LastName FROM Contact";
    try {
        QueryResult qr = connection.query(soqlQuery);
        boolean done = false;

        if (qr.getSize() > 0) {
            System.out.println("\nLogged-in user can see "
                + qr.getRecords().length + " contact records.");

            while (!done) {
                System.out.println("");
                SObject[] records = qr.getRecords();
                for (int i = 0; i < records.length; ++i) {
                    Contact con = (Contact) records[i];
                    String fName = con.getFirstName();
                    String lName = con.getLastName();

                    if (fName == null) {
                        System.out.println("Contact " + (i + 1) + ": " + lName);
                    } else {
                        System.out.println("Contact " + (i + 1) + ": " + fName
                            + " " + lName);
                    }
                }
            }

            if (qr.isDone()) {
                done = true;
            } else {

```

```

                qr = connection.queryMore(qr.getQueryLocator());
            }
        }
    } else {
        System.out.println("No records found.");
    }
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}
}

```

C# Sample Code

This section walks through a sample C# client application. The purpose of this sample application is to show the required steps for logging in and to demonstrate the invocation and subsequent handling of several API calls.

This sample application performs the following main tasks:

1. Prompts the user for their Salesforce username and password.
2. Calls `login()` to log in to the single login server and, if the login succeeds:
 - Sets the returned `sessionId` into the session header, which is required for session authentication on subsequent API calls.
 - Resets the Lightning Platform endpoint to the returned `serverUrl`, which is the target of subsequent API calls.

All client applications that access the API must complete the tasks in this step before attempting any subsequent API calls.

 - Retrieves user information and writes it to the console along with session information.
3. Calls `describeGlobal()` to retrieve a list of all available objects for the organization's data. The `describeGlobal` method determines [the objects that are available to the logged in user](#). This call should not be made more than once per session, since the data returned from the call is not likely to change frequently. The `DescribeGlobalResult` is echoed to the console.
4. Calls `describeSObjects()` to retrieve metadata (field list and object properties) for a specified object. The `describeSObject` method illustrates the type of metadata information that can be obtained for each object available to the user. The sample client application executes a `describeSObjects()` call on the object that the user specifies and then echoes the returned metadata information to the console. Object metadata information includes permissions, field types and lengths, and available values for picklist fields and types for `referenceTo` fields.
5. Calls `query()`, passing a simple query string (`"SELECT FirstName, LastName FROM Contact"`), and iterating through the returned [QueryResult](#).
6. Calls `logout()` to log the user out.

The following sample code uses try/catch blocks to handle exceptions that might be thrown by the API calls.

The following code begins the sample C# client application.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.Services.Protocols;
using Walkthrough.sforce;

namespace Walkthrough
{

```



```
class QuickstartApiSample
{
    private SforceService binding;

    [STAThread]
    static void Main(string[] args)
    {
        QuickstartApiSample sample = new QuickstartApiSample();
        sample.run();
    }

    public void run()
    {
        // Make a login call
        if (login())
        {
            // Do a describe global
            describeGlobalSample();

            // Describe an account object
            describeSObjectsSample();

            // Retrieve some data using a query
            querySample();

            // Log out
            logout();
        }
    }

    private bool login()
    {
        Console.WriteLine("Enter username: ");
        string username = Console.ReadLine();
        Console.WriteLine("Enter password: ");
        string password = Console.ReadLine();

        // Create a service object
        binding = new SforceService();

        // Timeout after a minute
        binding.Timeout = 60000;

        // Try logging in
        LoginResult lr;
        try
        {
            Console.WriteLine("\nLogging in...\n");
            lr = binding.login(username, password);
        }

        // ApiFault is a proxy stub generated from the WSDL contract when
```

```
// the web service was imported
catch (SoapException e)
{
    // Write the fault code to the console
    Console.WriteLine(e.Code);

    // Write the fault message to the console
    Console.WriteLine("An unexpected error has occurred: " + e.Message);

    // Write the stack trace to the console
    Console.WriteLine(e.StackTrace);

    // Return False to indicate that the login was not successful
    return false;
}

// Check if the password has expired
if (lr.passwordExpired)
{
    Console.WriteLine("An error has occurred. Your password has expired.");
    return false;
}

/** Once the client application has logged in successfully, it will use
 * the results of the login call to reset the endpoint of the service
 * to the virtual server instance that is servicing your organization
 */
// Save old authentication end point URL
String authEndPoint = binding.Url;
// Set returned service endpoint URL
binding.Url = lr.serverUrl;

/** The sample client application now has an instance of the SforceService
 * that is pointing to the correct endpoint. Next, the sample client
 * application sets a persistent SOAP header (to be included on all
 * subsequent calls that are made with SforceService) that contains the
 * valid sessionId for our login credentials. To do this, the sample
 * client application creates a new SessionHeader object and persist it to
 * the SforceService. Add the session ID returned from the login to the
 * session header
 */
binding.SessionHeaderValue = new SessionHeader();
binding.SessionHeaderValue.sessionId = lr.sessionId;

printUserInfo(lr, authEndPoint);

// Return true to indicate that we are logged in, pointed
// at the right URL and have our security token in place.
return true;
}
```

```
private void printUserInfo(LoginResult lr, String authEP)
{
    try
    {
        GetUserInfoResult userInfo = lr.userInfo;

        Console.WriteLine("\nLogging in ...\n");
        Console.WriteLine("UserID: " + userInfo.userId);
        Console.WriteLine("User Full Name: " +
            userInfo.userFullName);
        Console.WriteLine("User Email: " +
            userInfo.userEmail);
        Console.WriteLine();
        Console.WriteLine("SessionID: " +
            lr.sessionId);
        Console.WriteLine("Auth End Point: " +
            authEP);
        Console.WriteLine("Service End Point: " +
            lr.serverUrl);
        Console.WriteLine();
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " + e.Message +
            " Stack trace: " + e.StackTrace);
    }
}

private void logout()
{
    try
    {
        binding.logout();
        Console.WriteLine("Logged out.");
    }
    catch (SoapException e)
    {
        // Write the fault code to the console
        Console.WriteLine(e.Code);

        // Write the fault message to the console
        Console.WriteLine("An unexpected error has occurred: " + e.Message);

        // Write the stack trace to the console
        Console.WriteLine(e.StackTrace);
    }
}

/**
 * To determine the objects that are available to the logged-in
 * user, the sample client application executes a describeGlobal
 * call, which returns all of the objects that are visible to
 * the logged-in user. This call should not be made more than
 * once per session, as the data returned from the call likely
```

```

* does not change frequently. The DescribeGlobalResult is
* simply echoed to the console.
*/
private void describeGlobalSample()
{
    try
    {
        // describeGlobal() returns an array of object results that
        // includes the object names that are available to the logged-in user.
        DescribeGlobalResult dgr = binding.describeGlobal();

        Console.WriteLine("\nDescribe Global Results:\n");
        // Loop through the array echoing the object names to the console

        for (int i = 0; i < dgr.subjects.Length; i++)
        {
            Console.WriteLine(dgr.subjects[i].name);
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An exception has occurred: " + e.Message +
            "\nStack trace: " + e.StackTrace);
    }
}

/**
* The following method illustrates the type of metadata
* information that can be obtained for each object available
* to the user. The sample client application executes a
* describeSObject call on a given object and then echoes
* the returned metadata information to the console. Object
* metadata information includes permissions, field types
* and length and available values for picklist fields
* and types for referenceTo fields.
*/
private void describeSObjectsSample()
{
    Console.WriteLine("\nType the name of the object to " +
        "describe (try Account): ");
    string objectType = Console.ReadLine();
    try
    {
        // Call describeSObjects() passing in an array with one object type name
        DescribeSObjectResult[] dsrArray =
            binding.describeSObjects(new string[] { objectType });

        // Since we described only one sObject, we should have only
        // one element in the DescribeSObjectResult array.
        DescribeSObjectResult dsr = dsrArray[0];

        // First, get some object properties
        Console.WriteLine("\n\nObject Name: " + dsr.name);
    }
}

```

```
if (dsr.custom) Console.WriteLine("Custom Object");
if (dsr.label != null) Console.WriteLine("Label: " + dsr.label);

// Get the permissions on the object
if (dsr.createable) Console.WriteLine("Createable");
if (dsr.deletable) Console.WriteLine("Deleteable");
if (dsr.queryable) Console.WriteLine("Queryable");
if (dsr.replicateable) Console.WriteLine("Replicateable");
if (dsr.retrieveable) Console.WriteLine("Retrieveable");
if (dsr.searchable) Console.WriteLine("Searchable");
if (dsr.undeletable) Console.WriteLine("Undeleteable");
if (dsr.updateable) Console.WriteLine("Updateable");

Console.WriteLine("Number of fields: " + dsr.fields.Length);

// Now, retrieve metadata for each field
for (int i = 0; i < dsr.fields.Length; i++)
{
    // Get the field
    Field field = dsr.fields[i];

    // Write some field properties
    Console.WriteLine("Field name: " + field.name);
    Console.WriteLine("\tField Label: " + field.label);

    // This next property indicates that this
    // field is searched when using
    // the name search group in SOSL
    if (field.nameField)
        Console.WriteLine("\tThis is a name field.");

    if (field.restrictedPicklist)
        Console.WriteLine("This is a RESTRICTED picklist field.");

    Console.WriteLine("\tType is: " + field.type.ToString());

    if (field.length > 0)
        Console.WriteLine("\tLength: " + field.length);

    if (field.scale > 0)
        Console.WriteLine("\tScale: " + field.scale);

    if (field.precision > 0)
        Console.WriteLine("\tPrecision: " + field.precision);

    if (field.digits > 0)
        Console.WriteLine("\tDigits: " + field.digits);

    if (field.custom)
        Console.WriteLine("\tThis is a custom field.");

    // Write the permissions of this field
    if (field.nullable) Console.WriteLine("\tCan be nulled.");
}
```

```

        if (field.createable) Console.WriteLine("\tCreateable");
        if (field.filterable) Console.WriteLine("\tFilterable");
        if (field.updateable) Console.WriteLine("\tUpdateable");

        // If this is a picklist field, show the picklist values
        if (field.type.Equals(fieldType.picklist))
        {
            Console.WriteLine("\tPicklist Values");
            for (int j = 0; j < field.picklistValues.Length; j++)
                Console.WriteLine("\t\t" + field.picklistValues[j].value);
        }

        // If this is a foreign key field (reference),
        // show the values
        if (field.type.Equals(fieldType.reference))
        {
            Console.WriteLine("\tCan reference these objects:");
            for (int j = 0; j < field.referenceTo.Length; j++)
                Console.WriteLine("\t\t" + field.referenceTo[j]);
        }
        Console.WriteLine("");
    }
}
}
catch (SoapException e)
{
    Console.WriteLine("An exception has occurred: " + e.Message +
        "\nStack trace: " + e.StackTrace);
}
Console.WriteLine("Press ENTER to continue...");
Console.ReadLine();
}

private void querySample()
{
    String sqlQuery = "SELECT FirstName, LastName FROM Contact";
    try
    {
        QueryResult qr = binding.query(sqlQuery);
        bool done = false;

        if (qr.size > 0)
        {
            Console.WriteLine("Logged-in user can see "
                + qr.records.Length + " contact records.");

            while (!done)
            {
                Console.WriteLine("");
                sObject[] records = qr.records;
                for (int i = 0; i < records.Length; i++)
                {
                    Contact con = (Contact)records[i];
                    string fName = con.FirstName;
                    string lName = con.LastName;
                }
            }
        }
    }
}

```

```

        if (fName == null)
            Console.WriteLine("Contact " + (i + 1) + ": " + lName);
        else
            Console.WriteLine("Contact " + (i + 1) + ": " + fName
                + " " + lName);
    }

    if (qr.done)
    {
        done = true;
    }
    else
    {
        qr = binding.queryMore(qr.queryLocator);
    }
}
}
else
{
    Console.WriteLine("No records found.");
}
}
catch (Exception ex)
{
    Console.WriteLine("\nFailed to execute query successfully," +
        "error message was: \n{0}", ex.Message);
}
Console.WriteLine("\nPress ENTER to continue...");
Console.ReadLine();
}
}
}

```

The following C# example is the same as the previous C# example, except it uses .NET 3.0 SoapClient services instead of .NET 2.0 SforceService services.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.ServiceModel;
using Walkthrough.sforce;

namespace Walkthrough
{
    class QuickstartApiSample
    {
        private static SoapClient loginClient; // for login endpoint
        private static SoapClient client; // for API endpoint
        private static SessionHeader header;
        private static EndpointAddress endpoint;

        static void Main(string[] args)
    }
}

```

```
{
    QuickstartApiSample sample = new QuickstartApiSample();
    sample.run();
}

public void run()
{
    // Make a login call
    if (login())
    {
        // Do a describe global
        describeGlobalSample();

        // Describe an account object
        describeSObjectsSample();

        // Retrieve some data using a query
        querySample();

        // Log out
        logout();
    }
}

private bool login()
{
    Console.WriteLine("Enter username: ");
    string username = Console.ReadLine();
    Console.WriteLine("Enter password: ");
    string password = Console.ReadLine();

    // Create a SoapClient specifically for logging in
    loginClient = new SoapClient();

    // (combine pw and token if necessary)
    LoginResult lr;
    try
    {
        Console.WriteLine("\nLogging in...\n");
        lr = loginClient.login(null, username, password);
    }
    catch (Exception e)
    {
        // Write the fault message to the console
        Console.WriteLine("An unexpected error has occurred: " + e.Message);

        // Write the stack trace to the console
        Console.WriteLine(e.StackTrace);
        return false;
    }

    // Check if the password has expired
    if (lr.passwordExpired)
    {

```



```

        Console.WriteLine("An error has occurred. Your password has expired.");
        return false;
    }

    /** Once the client application has logged in successfully, it will use
     * the results of the login call to reset the endpoint of the service
     * to the virtual server instance that is servicing your organization
     */

    // On successful login, cache session info and API endpoint info
    endpoint = new EndpointAddress(lr.serverUrl);

    /** The sample client application now has a cached EndpointAddress
     * that is pointing to the correct endpoint. Next, the sample client
     * application sets a persistent SOAP header that contains the
     * valid sessionId for our login credentials. To do this, the sample
     * client application creates a new SessionHeader object. Add the session
     * ID returned from the login to the session header
     */
    header = new SessionHeader();
    header.sessionId = lr.sessionId;

    // Create and cache an API endpoint client
    client = new SoapClient("Soap", endpoint);

    printUserInfo(lr, lr.serverUrl);

    // Return true to indicate that we are logged in, pointed
    // at the right URL and have our security token in place.
    return true;
}

private void printUserInfo(LoginResult lr, String authEP)
{
    try
    {
        GetUserInfoResult userInfo = lr.userInfo;

        Console.WriteLine("\nLogging in ... \n");
        Console.WriteLine("UserID: " + userInfo.userId);
        Console.WriteLine("User Full Name: " +
            userInfo.userFullName);
        Console.WriteLine("User Email: " +
            userInfo.userEmail);
        Console.WriteLine();
        Console.WriteLine("SessionID: " +
            lr.sessionId);
        Console.WriteLine("Auth End Point: " +
            authEP);
        Console.WriteLine("Service End Point: " +
            lr.serverUrl);
        Console.WriteLine();
    }
    catch (Exception e)

```

```
    {
        Console.WriteLine("An unexpected error has occurred: " + e.Message +
            " Stack trace: " + e.StackTrace);
    }
}

private void logout()
{
    try
    {
        client.logout(header);
        Console.WriteLine("Logged out.");
    }
    catch (Exception e)
    {
        // Write the fault message to the console
        Console.WriteLine("An unexpected error has occurred: " + e.Message);

        // Write the stack trace to the console
        Console.WriteLine(e.StackTrace);
    }
}

/**
 * To determine the objects that are available to the logged-in
 * user, the sample client application executes a describeGlobal
 * call, which returns all of the objects that are visible to
 * the logged-in user. This call should not be made more than
 * once per session, as the data returned from the call likely
 * does not change frequently. The DescribeGlobalResult is
 * simply echoed to the console.
 */
private void describeGlobalSample()
{
    try
    {
        // describeGlobal() returns an array of object results that
        // includes the object names that are available to the logged-in user.
        DescribeGlobalResult dgr = client.describeGlobal(
            header, // session header
            null // package version header
        );

        Console.WriteLine("\nDescribe Global Results:\n");
        // Loop through the array echoing the object names to the console

        for (int i = 0; i < dgr.subjects.Length; i++)
        {
            Console.WriteLine(dgr.subjects[i].name);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("An exception has occurred: " + e.Message +
```

```

        "\nStack trace: " + e.StackTrace);
    }
}

/**
 * The following method illustrates the type of metadata
 * information that can be obtained for each object available
 * to the user. The sample client application executes a
 * describeSObject call on a given object and then echoes
 * the returned metadata information to the console. Object
 * metadata information includes permissions, field types
 * and length and available values for picklist fields
 * and types for referenceTo fields.
 */
private void describeSObjectsSample()
{
    Console.WriteLine("\nType the name of the object to " +
        "describe (try Account): ");
    string objectType = Console.ReadLine();
    try
    {
        // Call describeSObjects() passing in an array with one object type name

        DescribeSObjectResult[] dsrArray =
            client.describeSObjects(
                header, // session header
                null, // package version header
                null, // locale options
                new string[] { objectType } // object name array
            );

        // Since we described only one sObject, we should have only
        // one element in the DescribeSObjectResult array.
        DescribeSObjectResult dsr = dsrArray[0];

        // First, get some object properties
        Console.WriteLine("\n\nObject Name: " + dsr.name);

        if (dsr.custom) Console.WriteLine("Custom Object");
        if (dsr.label != null) Console.WriteLine("Label: " + dsr.label);

        // Get the permissions on the object
        if (dsr.createable) Console.WriteLine("Createable");
        if (dsr.deletable) Console.WriteLine("Deleteable");
        if (dsr.queryable) Console.WriteLine("Queryable");
        if (dsr.replicateable) Console.WriteLine("Replicateable");
        if (dsr.retrieveable) Console.WriteLine("Retrieveable");
        if (dsr.searchable) Console.WriteLine("Searchable");
        if (dsr.undeletable) Console.WriteLine("Undeleteable");
        if (dsr.updateable) Console.WriteLine("Updateable");

        Console.WriteLine("Number of fields: " + dsr.fields.Length);
    }
}

```

```

// Now, retrieve metadata for each field
for (int i = 0; i < dsr.fields.Length; i++)
{
    // Get the field
    Field field = dsr.fields[i];

    // Write some field properties
    Console.WriteLine("Field name: " + field.name);
    Console.WriteLine("\tField Label: " + field.label);

    // This next property indicates that this
    // field is searched when using
    // the name search group in SOSL
    if (field.nameField)
        Console.WriteLine("\tThis is a name field.");

    if (field.restrictedPicklist)
        Console.WriteLine("This is a RESTRICTED picklist field.");

    Console.WriteLine("\tType is: " + field.type.ToString());

    if (field.length > 0)
        Console.WriteLine("\tLength: " + field.length);

    if (field.scale > 0)
        Console.WriteLine("\tScale: " + field.scale);

    if (field.precision > 0)
        Console.WriteLine("\tPrecision: " + field.precision);

    if (field.digits > 0)
        Console.WriteLine("\tDigits: " + field.digits);

    if (field.custom)
        Console.WriteLine("\tThis is a custom field.");

    // Write the permissions of this field
    if (field.nillable) Console.WriteLine("\tCan be nulled.");
    if (field.createable) Console.WriteLine("\tCreateable");
    if (field.filterable) Console.WriteLine("\tFilterable");
    if (field.updateable) Console.WriteLine("\tUpdateable");

    // If this is a picklist field, show the picklist values
    if (field.type.Equals(fieldType.picklist))
    {
        Console.WriteLine("\tPicklist Values");
        for (int j = 0; j < field.picklistValues.Length; j++)
            Console.WriteLine("\t\t" + field.picklistValues[j].value);
    }

    // If this is a foreign key field (reference),
    // show the values
    if (field.type.Equals(fieldType.reference))
    {

```

```

        Console.WriteLine("\tCan reference these objects:");
        for (int j = 0; j < field.referenceTo.Length; j++)
            Console.WriteLine("\t\t" + field.referenceTo[j]);
    }
    Console.WriteLine("");
}
}
catch (Exception e)
{
    Console.WriteLine("An exception has occurred: " + e.Message +
        "\nStack trace: " + e.StackTrace);
}
Console.WriteLine("Press ENTER to continue...");
Console.ReadLine();
}

private void querySample()
{
    String sqlQuery = "SELECT FirstName, LastName FROM Contact";
    try
    {
        QueryResult qr = client.query(
            header, // session header
            null, // query options
            null, // mru options
            null, // package version header
            sqlQuery // query string
        );

        bool done = false;

        if (qr.size > 0)
        {
            Console.WriteLine("Logged-in user can see "
                + qr.records.Length + " contact records.");

            while (!done)
            {
                Console.WriteLine("");
                sObject[] records = qr.records;
                for (int i = 0; i < records.Length; i++)
                {
                    Contact con = (Contact)records[i];
                    string fName = con.FirstName;
                    string lName = con.LastName;
                    if (fName == null)
                        Console.WriteLine("Contact " + (i + 1) + ": " + lName);
                    else
                        Console.WriteLine("Contact " + (i + 1) + ": " + fName
                            + " " + lName);
                }

                if (qr.done)
                {

```

```
        done = true;
    }
    else
    {
        qr = client.queryMore(
            header, // session header
            null, // query options
            qr.queryLocator // query locator
        );
    }
}
else
{
    Console.WriteLine("No records found.");
}
}
catch (Exception ex)
{
    Console.WriteLine("\nFailed to execute query successfully," +
        "error message was: \n{0}", ex.Message);
}
Console.WriteLine("\nPress ENTER to continue...");
Console.ReadLine();
}
}
```

CHAPTER 2 Core Data Types Used in API Calls

In this chapter ...

- [sObject](#)
- [API Fault Element](#)
- [ExceptionCode](#)
- [Error](#)
- [StatusCode](#)
- [ExtendedErrorDetails](#)
- [ExtendedErrorCode](#)
- [Duplicate Management Data Types](#)

Many calls in the API use the following data types:

- [sObject](#)
- ID (String). See [ID Field Types](#).

The API also uses several error handling objects. If an error occurs during a SOAP request, the API returns a SOAP fault message. The message contains different content, depending on the type of error:

- If an error affects the entire request, an [API Fault Element](#), is returned, containing an [ExceptionCode](#) and the associated error message text.
- If the error affects some records and not others, an [Error](#) is returned, containing a [StatusCode](#). These errors typically occur during bulk operations, such as creating, updating, or deleting multiple records with a single call.

You can see the list of exception codes, status codes, and extended error codes in the WSDL file for your org. Some codes don't appear in your WSDL, depending on the features enabled. See [Generating the WSDL File for Your Organization](#).

sObject

An sObject represents an object, such as an [Account](#) or [Campaign](#). For a list of standard objects, see [Standard Objects](#).

An sObject has the following properties:

Name	Type	Description
<code>fieldsToNull</code>	<code>string[]</code>	Array of one or more field names whose value you want to explicitly set to <code>null</code> . When used with <code>update()</code> or <code>upsert()</code> , you can specify only those fields that you can update and that have the <code>nullable</code> property. When used with <code>create()</code> , you can specify only those fields that you can create and that have the <code>nullable</code> or the <code>default on create</code> property. For example, if specifying an ID field or required field results in a runtime error, you can specify that field name in <code>fieldsToNull</code> . Similarly, if a picklist field has a default value and you want to set the value to <code>null</code> instead, specify the field in <code>fieldsToNull</code> .
<code>ID</code>	<code>ID</code>	Unique ID for this individual object. For the <code>create()</code> call, this value is <code>null</code> . For all other API calls, this value must be specified.

API Fault Element

An `ApiFault` element contains information about a fault that occurs when processing a service request. The `ApiFault` element has the following properties.

Name	Type	Description
<code>exceptionCode</code>	ExceptionCode	A code that characterizes the exception.
<code>exceptionMessage</code>	<code>string</code>	Exception message text.
<code>extendedErrorDetails</code>	ExtendedErrorDetails	Additional details about the exception, including an extended error code and extra error properties, when available.

The following table lists the API fault elements that represent all the API faults that can occur.

Fault	Description
<code>ApiQueryFault</code>	The row and column numbers where the problem occurred.
<code>LoginFault</code>	An error occurred during the <code>login()</code> call.
<code>InvalidSObjectFault</code>	An invalid sObject in a <code>describeObject()</code> , <code>describeSObjects()</code> , <code>describeLayout()</code> , <code>describeDataCategoryGroups()</code> , <code>describeDataCategoryGroupStructures()</code> , <code>create()</code> , <code>update()</code> , <code>retrieve()</code> , or <code>query()</code> call.
<code>InvalidFieldFault</code>	An invalid field in a <code>retrieve()</code> or <code>query()</code> call.
<code>InvalidOrNullForRestrictedPicklist</code>	An invalid <code>appMenuItem</code> in a <code>describeAppMenu()</code> call.

Fault	Description
MalformedQueryFault	A problem in the <code>queryString</code> passed in a <code>query()</code> call.
InvalidQueryLocatorFault	A problem in the <code>queryLocator</code> passed in a <code>queryMore()</code> call.
MalformedSearchFault	A problem in the <code>search</code> passed in a <code>search()</code> call.
InvalidIdFault	A specified ID was invalid in a <code>setPassword()</code> or <code>resetPassword()</code> call.
UnexpectedErrorFault	An unexpected error occurred. The error isn't associated with any other API fault.

ExceptionCode

The following exception codes can be returned with an error.

APEX_REST_SERVICES_DISABLED

Apex REST Services permission hasn't been enabled for the user. Enable the user permission to access Apex classes and methods as REST web services.

API_CURRENTLY_DISABLED

Because of a system problem, API functionality is temporarily unavailable.

API_DISABLED_FOR_ORG

API access hasn't been enabled for the org. Contact Salesforce to enable API access.

CANT_ADD_STANDARD_PORTAL_USER_TO_TERRITORY

A user with a standard portal license can't be added to a territory.

CIRCULAR_OBJECT_GRAPH

The request failed because it contained a circular object reference.

CLIENT_NOT_ACCESSIBLE_FOR_USER

The current user doesn't have permission to access the specified client.

CLIENT_REQUIRE_UPDATE_FOR_USER

The current user is required to use a newer version of the specified client and doesn't have access until the client is updated.

CLONE_NOT_SUPPORTED

This entity doesn't support the clone operation.

CLONE_FIELD_INTEGRITY_EXCEPTION

A field integrity exception occurred during the clone operation.

CONTENT_ALREADY_AN_ASSET_EXCEPTION

File with id: {id} is already an asset.

CONTENT_HUB_AUTHENTICATION_EXCEPTION

Authentication token has expired.

CONTENT_HUB_FILE_HAS_NO_URL_EXCEPTION

Can't open file.

CONTENT_HUB_FILE_NOT_FOUND_EXCEPTION

Object not found.

CONTENT_HUB_INVALID_PAGE_NUMBER_EXCEPTION

This document has no content.

CONTENT_HUB_INVALID_OBJECT_TYPE_EXCEPTION

Invalid object type.

CONTENT_HUB_INVALID_RENDITION_PAGE_NUMBER_EXCEPTION

Not a valid rendition page number.

CONTENT_HUB_ITEM_TYPE_NOT_FOUND_EXCEPTION

Item type not found.

CONTENT_HUB_OBJECT_NOT_FOUND_EXCEPTION

Object not found.

CONTENT_HUB_OPERATION_NOT_SUPPORTED_EXCEPTION

Operation not supported.

CONTENT_HUB_SECURITY_EXCEPTION

Unauthorized operation.

CONTENT_HUB_TIMEOUT_EXCEPTION

Operation timed out.

CONTENT_HUB_UNEXPECTED_EXCEPTION

An error occurred while performing this operation.

CONTENT_IMAGE_SCALING_INVALID_ARGUMENTS_EXCEPTION

Invalid argument type for the operation.

CONTENT_IMAGE_SCALING_INVALID_IMAGE_EXCEPTION

The specified image isn't valid.

CONTENT_IMAGE_SCALING_MAX_RENDITIONS_EXCEPTION

You've reached the maximum number of renditions for the image.

CONTENT_IMAGE_SCALING_TIMEOUT_EXCEPTION

The image scaling operation timed out.

CONTENT_IMAGE_SCALING_UNKNOWN_EXCEPTION

The system encountered an internal error during image scaling. Report this problem to Salesforce.

DELETE_REQUIRED_ON_CASCADE

The delete operation triggers a cascade delete on a record, but the logged-in user doesn't have delete permission on that related object.

DUPLICATE_COMM_NICKNAME

You can't create a user with the same nickname as another user.

DUPLICATE_VALUE

You can't supply a duplicate value for a field that must be unique. For example, you can't submit two copies of the same session ID in a `invalidateSessions()` call.

EMAIL_BATCH_SIZE_LIMIT_EXCEEDED

A method tried to process more email records than the maximum batch size.

EMAIL_TO_CASE_INVALID_ROUTING

An Email-to-Case record has been submitted for processing but the feature isn't enabled.

EMAIL_TO_CASE_LIMIT_EXCEEDED

The daily converted email limit for the Email-to-Case feature has been exceeded.

EMAIL_TO_CASE_NOT_ENABLED

The Email-to-Case feature hasn't been enabled.

EXCEEDED_ID_LIMIT

Too many IDs were specified in a call. For example, more than 2000 IDs were requested in a `retrieve()` call, or more than 200 session IDs were specified in a `logout()` call.

EXCEEDED_LEAD_CONVERT_LIMIT

Too many IDs were sent to a `convertLead()` call.

EXCEEDED_MAX_SEMIJOIN_SUBSELECTS

Too many topic filters were applied to a list view.

EXCEEDED_MAX_SIZE_REQUEST

The size of the message sent to the API exceeded 50 MB.

EXCEEDED_MAX_TYPES_LIMIT

The number of object types to describe is too large.

EXCEEDED_QUOTA

The size limit for org data storage was exceeded during a `create()` call.

FUNCTIONALITY_NOT_ENABLED

Functionality has been temporarily disabled. Other calls continue to work.

GONE

The requested resource or operation has been retired or removed. Delete or update any references to the resource or operation.

INACTIVE_OWNER_OR_USER

The user or record owner isn't active.

INACTIVE_PORTAL

The referenced portal is inactive.

INSUFFICIENT_ACCESS

The user doesn't have sufficient access to perform the operation.

INVALID_ASSIGNMENT_RULE

An invalid `AssignmentRuleHeader` value was specified.

INVALID_BATCH_SIZE

The query options have an invalid batch size value.

INVALID_CLIENT

The client is invalid.

INVALID_CROSS_REFERENCE_KEY

An invalid foreign key can't be set on a field. For example, an object share, such as `AccountShare`, can't be deleted because the share row is a result of a sharing rule.

INVALID_FIELD

The specified field name is invalid.

INVALID_FILTER_LANGUAGE

The specified language can't be used as a filter.

INVALID_FILTER_VALUE

A SOQL query with `LIKE` specified an invalid character, for example, an incorrectly placed asterisk (*). Correct the query and resubmit.

INVALID_ID_FIELD

The specified ID is correctly formatted but isn't valid. For example, the ID is of the wrong type, or the object it identifies no longer exists.

INVALID_GOOGLE_DOCS_URL

An invalid Salesforce record URL was used when trying to associate a Google Doc to that record. Correct the URL before trying the operation again.

INVALID_LOCATOR

The locator is invalid.

INVALID_LOGIN

The `login()` credentials aren't valid, or the maximum number of logins have been exceeded. Contact your administrator for more information.

INVALID_NEW_PASSWORD

The new password doesn't conform with the password policies of the org.

INVALID_OPERATION

The client application tried to modify a record that is locked by an approval process.

INVALID_OPERATION_WITH_EXPIRED_PASSWORD

Due to password expiration, a valid password must be set using `setPassword()` before the call can be invoked.

INVALID_QUERY_FILTER_OPERATOR

An invalid operator was used in the `query()` filter clause, at least for that field.

INVALID_QUERY_LOCATOR

An invalid `queryLocator` parameter was specified in a `queryMore()` call. It's also possible that you've exceeded the maximum number of calls, which is 10 per user for the API, and 5 for Apex and Visualforce.

INVALID_QUERY_SCOPE

The specified search scope is invalid.

INVALID_REPLICATION_DATE

The date for replication is out of the allowed range, such as before the org was created.

INVALID_SETUP_OWNER

The setup owner must be an Organization, Profile, or User.

INVALID_SEARCH

The `search()` call has invalid syntax or grammar. For more information, see the *Salesforce SOQL and SOSL Reference Guide*.

INVALID_SEARCH_SCOPE

The specified search scope is invalid.

INVALID_SESSION_ID

The specified `sessionId` is malformed (incorrect length or format) or has expired. Log in again to start a new session.

INVALID_SOAP_HEADER

There's an error in the SOAP header. If you're migrating from an earlier version of the API, be advised that the `SaveOptions` header can't be used with API version 6.0 or later. Use `AssignmentRuleHeader` instead.

INVALID_SSO_GATEWAY_URL

The URL provided to configure the Single Sign-On gateway wasn't a valid URL.

INVALID_TYPE

The specified `sObject` type invalid.

INVALID_TYPE_FOR_OPERATION

The specified [sObject](#) type is invalid for the specified operation.

INVALID_VERSION

The requested resource no longer exists. Confirm the API version number used in the request is supported, and update if needed.

LIMIT_EXCEEDED

An array is too long. For example, there are too many BCC addresses, targets, or email messages.

LOGIN_DURING_RESTRICTED_DOMAIN

The user isn't allowed to log in from this IP address.

LOGIN_DURING_RESTRICTED_TIME

The user isn't allowed to log in during this time period.

MALFORMED_ID

An invalid ID string was specified. For information about IDs, see [ID Field Type](#).

MALFORMED_QUERY

An invalid query string was specified. For example, the query string was longer than 100,000 characters.

MALFORMED_SEARCH

An invalid search string was specified. For example, the search string was longer than 100,000 characters.

MISSING_ARGUMENT

A required argument is missing.

MIXED_DML_OPERATION

There are limits on what kinds of DML operations can be performed in the same transaction. For more information, see [Data Manipulation Language](#) in the *Apex Developer Guide*.

NOT_MODIFIED

The describe call response hasn't changed since the specified date.

NO_SOFTPHONE_LAYOUT

If an org has the CTI feature enabled, but no softphone layout has been defined, this exception is returned if a describe call is issued. This exception is most often caused because no call center has been defined. A default softphone layout is created during call center definition.

If an org doesn't have the CTI feature enabled, `FUNCTIONALITY_NOT_ENABLED` is returned instead.

NUMBER_OUTSIDE_VALID_RANGE

The number specified is outside the valid range for the field.

OPERATION_TOO_LARGE

The query has returned too many results. If certain queries are run by a user without the "View All Data" permission and many records are returned, the queries require sharing rule checking. For example, consider queries that are run on objects, such as [Task](#), that use a polymorphic foreign key. These queries return this exception because the operation requires too many resources. To correct, add filters to the query to narrow the scope, or use filters such as date ranges to break the query up into a series of smaller queries.

ORDER_MANAGEMENT_ACTION_NOT_ALLOWED

The requested action isn't allowed.

ORG_LOCKED

The org has been locked. Contact Salesforce to unlock the org.

ORG_NOT_OWNED_BY_INSTANCE

The user tried to log in to the wrong server instance. Choose another server instance, use your org's My Domain login URL, or log in at <https://login.salesforce.com>.

PASSWORD_LOCKOUT

The user has exceeded the allowed number of login attempts. The user must contact an administrator to regain login access.

PORTAL_NO_ACCESS

Access to the specified portal isn't available.

QUERY_TIMEOUT

The query has timed out. For more information on query limits and how to avoid them, see [SOQL and SOSL Limits for Search Queries](#) in the *Developer Limits and Allocations Quick Reference* and [SOQL SELECT Syntax](#) in the *Salesforce SOQL and SOSL Reference Guide*.

QUERY_TOO_COMPLICATED

SOQL query is either selecting too many fields or there are too many filter conditions. Try reducing the number of formula fields referenced in the query.

REQUEST_LIMIT_EXCEEDED

Exceeded either the concurrent request limit or the request rate limit for your org. For details on API request limits, see [API Usage Metering](#).

REQUEST_RUNNING_TOO_LONG

A request has taken too long to be processed.

SERVER_UNAVAILABLE

A server that is necessary for this call is unavailable. Other types of requests could still work.

SSO_SERVICE_DOWN

The service was unavailable, and an authentication call to the org's specified Single Sign-On server failed.

TOO_MANY_APEX_REQUESTS

Too many Apex requests have been issued. If this exception persists, contact Salesforce Customer Support.

TRIAL_EXPIRED

The trial period for the org has expired. A representative from the company must contact Salesforce to re-enable the org.

UNSUPPORTED_API_VERSION

A method call was made that doesn't exist in the accessed API version, for example, trying to use [upsert\(\)](#) (new in 8.0) against version 5.0.


UNSUPPORTED_CLIENT

This version of the client is no longer supported.

Error

An `Error` contains information about an error that occurred during a [create\(\)](#), [merge\(\)](#), [process\(\)](#), [update\(\)](#), [upsert\(\)](#), [delete\(\)](#), or [undelete\(\)](#) call. For more information, see [Error Handling](#). An `Error` has the following properties:

Name	Type	Description
<code>statusCode</code>	StatusCode	A code that describes the error.
<code>message</code>	string	Error message text.
<code>fields</code>	string[]	Array of one or more field names. Identifies which fields in the object, if any, affected the error condition.
<code>extendedErrorDetails</code>	ExtendedErrorDetails	More details about the error, including an extended error code and extra error properties, when available.

 **Note:** If your org has active duplicate rules and a duplicate is detected, the SaveResult includes an error with a data type of DuplicateError.

StatusCode

An error can return any of these API status codes.

ALL_OR_NONE_OPERATION_ROLLED_BACK

The bulk operation was rolled back because one of the records wasn't processed successfully. See [AllOrNoneHeader](#).

ALREADY_IN_PROCESS

You can't submit a record that is already in an approval process. Wait for the previous approval process to complete before resubmitting a request with this record.

ASSIGNEE_TYPE_REQUIRED

Designate an assignee for the approval request (ProcessInstanceStep or ProcessInstanceWorkitem).

BAD_CUSTOM_ENTITY_PARENT_DOMAIN

The changes you're trying to make can't be completed because changes to the associated master-detail relationship can't be made.

BCC_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED

Your client application blind carbon-copied an email address even though the org's Compliance BCC Email option is enabled. This option specifies a particular email address that automatically receives a copy of all outgoing email. When this option is enabled, you can't BCC any other email address. To disable the option, log in to the user interface and from Setup, enter *Compliance BCC Email* in the Quick Find box, then select **Compliance BCC Email**.

BCC_SELF_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED

Your client application blind carbon-copied the logged-in user's email address even though the org's BCC COMPLIANCE option is set to true. This option specifies a particular email address that automatically receives a copy of all outgoing email. When this option is enabled, you can't BCC any other email address. To disable the option, log in to the user interface and from Setup, enter *Compliance BCC Email* in the Quick Find box, then select **Compliance BCC Email**.

CANNOT_CASCADE_PRODUCT_ACTIVE

An update to a product caused by a cascade can't be done because the associated product is active.

CANNOT_CHANGE_FIELD_TYPE_OF_APEX_REFERENCED_FIELD

You can't change the type of a field that is referenced in an Apex script.

CANNOT_CREATE_ANOTHER_MANAGED_PACKAGE

You can create only one managed package in an org.

CANNOT_DEACTIVATE_DIVISION

You can't deactivate Divisions if an assignment rule references divisions or if the `DefaultDivision` field on a user record isn't set to null.

CANNOT_DELETE_LAST_DATED_CONVERSION_RATE

If dated conversions are enabled, you must have at least one DatedConversionRate record.

CANNOT_DELETE_MANAGED_OBJECT

You can't modify components that are included in a managed package.

CANNOT_DISABLE_LAST_ADMIN

You must have at least one active administrator user.

CANNOT_ENABLE_IP_RESTRICT_REQUESTS

If you exceed the limit of five IP ranges specified in a profile, you can't enable restriction of login by IP addresses. Reduce the number of specified ranges in the profile and try the request again.

CANNOT_INSERT_UPDATE_ACTIVATE_ENTITY

You don't have permission to create, update, or activate the specified record.

CANNOT_MODIFY_MANAGED_OBJECT

You can't modify components that are included in a managed package.

CANNOT_RENAME_APEX_REFERENCED_FIELD

You can't rename a field that is referenced in an Apex script.

CANNOT_RENAME_APEX_REFERENCED_OBJECT

You can't rename an object that is referenced in an Apex script.

CANNOT_REPARENT_RECORD

You can't define a new parent record for the specified record.

CANNOT_RESOLVE_NAME

A `sendEmail()` call couldn't resolve an object name.

CANNOT_UPDATE_CONVERTED_LEAD

A converted lead couldn't be updated.

CANT_DISABLE_CORP_CURRENCY

You can't disable the corporate currency for an org. To disable a currency that is set as the corporate currency, first use the user interface to change the corporate currency to a different currency. Then disable the original currency.

CANT_UNSET_CORP_CURRENCY

You can't change the corporate currency for an org from the API. Use the user interface to change the corporate currency.

CHILD_SHARE_FAILS_PARENT

If you don't have appropriate permissions on a parent record, you can't change the owner of or define sharing rules for a child record. For example, you can't change the owner of a contact record if you can't edit its parent account record.

CIRCULAR_DEPENDENCY

You can't create a circular dependency between metadata objects in your org. For example, public group A can't include public group B, if public group B already includes public group A.

COMMUNITY_NOT_ACCESSIBLE

You don't have permission to access the Experience Cloud site that this entity belongs to. You must be given permission to access the site before you can access this entity.

CUSTOM_CLOB_FIELD_LIMIT_EXCEEDED

You can't exceed the maximum size for a CLOB field.

CUSTOM_ENTITY_OR_FIELD_LIMIT

You've reached the maximum number of custom objects or custom fields for your org.

CUSTOM_FIELD_INDEX_LIMIT_EXCEEDED

You've reached the maximum number of indexes on a field for your org.

CUSTOM_INDEX_EXISTS

You can create only one custom index per field.

CUSTOM_LINK_LIMIT_EXCEEDED

You've reached the maximum number of custom links for your org.

CUSTOM_METADATA_LIMIT_EXCEEDED

Your org has reached its custom metadata maximum limit.

CUSTOM_SETTINGS_LIMIT_EXCEEDED

Your org has reached its custom settings maximum limit.

CUSTOM_TAB_LIMIT_EXCEEDED

You've reached the maximum number of custom tabs for your org.

DELETE_FAILED

You can't delete a record because it is in use by another object.

DEPENDENCY_EXISTS

You can't perform the requested operation because of an existing dependency on the specified object or field.

DUPLICATE_CASE_SOLUTION

You can't create a relationship between the specified case and solution because it already exists.

DUPLICATE_CUSTOM_ENTITY_DEFINITION

Custom object or custom field IDs must be unique.

DUPLICATE_CUSTOM_TAB_MOTIF

You can't create a custom object or custom field with a duplicate master name.

DUPLICATE_DEVELOPER_NAME

You can't create a custom object or custom field with a duplicate developer name.

DUPLICATES_DETECTED

Duplicate records have been detected. Used for an Error object with a data type of [DuplicateError](#).

DUPLICATE_EXTERNAL_ID

A user-specified external ID matches more than one record during an upsert.

DUPLICATE_MASTER_LABEL

You can't create a custom object or custom field with a duplicate master name.

DUPLICATE_SENDER_DISPLAY_NAME

A `sendEmail()` call couldn't choose between `OrgWideEmailAddress.DisplayName` or `senderDisplayName`. Define only one of the two fields.

DUPLICATE_USERNAME

A create, update, or upsert failed because of a duplicate username.

DUPLICATE_VALUE

You can't supply a duplicate value for a field that must be unique. For example, you can't submit two copies of the same session ID in a `invalidateSessions()` call.

EMAIL_ADDRESS_BOUNCED

Emails to one or more recipients have bounced. Check the email addresses to make sure that they're valid.

EMAIL_NOT_PROCESSED_DUE_TO_PRIOR_ERROR

Because of an error earlier in the call, the current email wasn't processed.

EMAIL_OPTED_OUT

A single email message was sent with the `REJECT` setting in the `optOutPolicy` field to recipients that have opted out from receiving email. To avoid this error, set the `optOutPolicy` field to another value.

EMAIL_TEMPLATE_FORMULA_ERROR

The email template is invalid and can't be rendered. Check the template for incorrectly specified merge fields.

EMAIL_TEMPLATE_MERGEFIELD_ACCESS_ERROR

You don't have access to one or more merge fields in this template. To request access, contact your Salesforce administrator.

EMAIL_TEMPLATE_MERGEFIELD_ERROR

One or more merge fields don't exist. Check the spelling of field names.

EMAIL_TEMPLATE_MERGEFIELD_VALUE_ERROR

One or more merge fields have no value. To provide values, update the records before sending the email.

EMAIL_TEMPLATE_PROCESSING_ERROR

The merge fields in this email template can't be processed. Ensure that your template body is valid.

EMPTY_SCONTROL_FILE_NAME

The Scontrol file name was empty, but the binary wasn't empty.

ENTITY_FAILED_IFLASTMODIFIED_ON_UPDATE

If the value in a record's `LastModifiedDate` field is later than the current date, you can't update the record.

ENTITY_IS_ARCHIVED

If a record has been archived, you can't access it.

ENTITY_IS_DELETED

You can't reference an object that has been deleted. This status code occurs only in API version 10.0 and later. Previous releases of the API use `INVALID_ID_FIELD` for this error.

ENTITY_IS_LOCKED

You can't edit a record that is locked by an approval process.

ENVIRONMENT_HUB_MEMBERSHIP_CONFLICT

You can't add an org to more than one Environment Hub.

ERROR_IN_MAILER

An email address is invalid, or another error occurred during an email-related transaction.

FAILED_ACTIVATION

The activation of a Contract failed.

FIELD_CUSTOM_VALIDATION_EXCEPTION

You can't define a custom validation formula that violates a field integrity rule.

FIELD_FILTER_VALIDATION_EXCEPTION

You can't violate field integrity rules.

FILTERED_LOOKUP_LIMIT_EXCEEDED

The creation of the lookup filter failed because it exceeds the maximum number of lookup filters allowed per object.

HTML_FILE_UPLOAD_NOT_ALLOWED

Your attempt to upload an HTML file failed. HTML attachments and documents, including HTML attachments to a [Solution](#), can't be uploaded if the `Disallow HTML documents and attachments` checkbox is selected on the HTML Documents and Attachments Settings page.

IMAGE_TOO_LARGE

The image exceeds the maximum width, height, and file size.

INACTIVE_OWNER_OR_USER

The owner of the specified item is an inactive user. To reference this item, either reactivate the owner or reassign ownership to another active user.

INSERT_UPDATE_DELETE_NOT_ALLOWED_DURING_MAINTENANCE

Starting with version 32.0, you can't create, update, or delete data while the instance where your org resides is being upgraded to the latest release. Try again after the release has completed. For release schedules, see trust.salesforce.com. Before version 32.0, the code is `INVALID_READ_ONLY_USER_DML`.

INSUFFICIENT_ACCESS_ON_CROSS_REFERENCE_ENTITY

An operation affects an object that is cross-referenced by the specified object, but the logged-in user doesn't have sufficient permissions on the cross-referenced object. For example, a logged-in user attempts to modify an account record, and the update

creates a `ProcessInstanceWorkitem`. If the user doesn't have permission to approve, reject, or reassign the `ProcessInstanceWorkitem`, this exception occurs.

INSUFFICIENT_ACCESS_OR_READONLY

You can't perform the specified action because you don't have sufficient permissions.

INVALID_ACCESS_LEVEL

You can't define a new sharing rule that provides less access than the specified org-wide default.

INVALID_ARGUMENT_TYPE

You supplied an argument that is of the wrong type for the operation being attempted.

INVALID_ASSIGNEE_TYPE

You specified an assignee type that isn't a valid integer between one and six.

INVALID_ASSIGNMENT_RULE

You specified an assignment rule that is invalid or that isn't defined in the org.

INVALID_BATCH_OPERATION

The specified batch operation is invalid.

INVALID_CONTENT_TYPE

The outgoing email has an `EmailFileAttachment` with an invalid `contentType` property. See [RFC2045 - Internet Message Format](#).

INVALID_CREDIT_CARD_INFO

The specified credit card information isn't valid.

INVALID_CROSS_REFERENCE_KEY

The specified value in a relationship field is not valid, or data is the expected type.

INVALID_CROSS_REFERENCE_TYPE_FOR_FIELD

The specified cross-reference type isn't valid for the specified field.

INVALID_CURRENCY_CONV_RATE

Specify a positive, non-zero value for the currency conversion rate.

INVALID_CURRENCY_CORP_RATE

You can't modify the corporate currency conversion rate.

INVALID_CURRENCY_ISO

The specified [currency ISO code](#) isn't valid.

INVALID_EMAIL_ADDRESS

A specified email address is invalid.

INVALID_EMPTY_KEY_OWNER

You can't set the value for `owner` to null.

INVALID_EVENT_SUBSCRIPTION

Invalid parameters were specified when subscribing to an event.

INVALID_FIELD

You specified an invalid field name when trying to update or upsert a record.

INVALID_FIELD_FOR_INSERT_UPDATE

You can't combine a person account record type change with any other field update.

INVALID_FIELD_WHEN_USING_TEMPLATE

You can't use an email template with an invalid field name.

INVALID_FILTER_ACTION

The specified filter action can't be used with the specified object. For example, an alert isn't a valid filter action for a Task.

INVALID_ID_FIELD

The specified ID field (ID, ownerId), or cross-reference field is invalid.

INVALID_INET_ADDRESS

A specified Inet address isn't valid.

INVALID_LINEITEM_CLONE_STATE

You can't clone a Pricebook2 or PricebookEntry record that isn't active.

INVALID_MASTER_OR_TRANSLATED_SOLUTION

The solution is invalid. For example, this exception occurs if you try to associate a translated solution with a master solution that's associated with another translated solution.

INVALID_MESSAGE_ID_REFERENCE

The outgoing email's References or In-Reply-To fields are invalid. These fields must contain valid Message-IDs. See [RFC2822 - Internet Message Format](#).

INVALID_OPERATION

There's no applicable approval process for the specified object.

INVALID_OPERATOR

The specified operator isn't applicable for the field type when used as a workflow filter.

INVALID_OR_NULL_FOR_RESTRICTED_PICKLIST

You specified an invalid or null value for a restricted picklist.

INVALID_PARTNER_NETWORK_STATUS

The specified partner network status is invalid for the specified template field.

INVALID_PERSON_ACCOUNT_OPERATION

You can't delete a person account.

INVALID_READ_ONLY_USER_DML

Version 31.0 and earlier: You can't create, update, or delete data while the instance where your org resides is being upgraded to the latest release. Try again after the release has completed. For release schedules, see trust.salesforce.com. After version 31.0, the code is `INSERT_UPDATE_DELETE_NOT_ALLOWED_DURING_MAINTENANCE`.

INVALID_SAVE_AS_ACTIVITY_FLAG

Specify `true` or `false` for the `saveAsActivity` flag.

INVALID_SESSION_ID

The specified `sessionId` is malformed (incorrect length or format) or has expired. Log in again to start a new session.

INVALID_STATUS

The specified org status change isn't valid.

INVALID_TYPE

The specified type isn't valid for the specified object.

INVALID_TYPE_FOR_OPERATION

The specified type isn't valid for the specified operation.

INVALID_TYPE_ON_FIELD_IN_RECORD

The specified value isn't valid for the specified field's type.

IP_RANGE_LIMIT_EXCEEDED

The specified IP address is outside the IP range specified for the org.

JIGSAW_IMPORT_LIMIT_EXCEEDED

The number of records you attempted to purchase from Data.com exceeds your available record addition limit.

LICENSE_LIMIT_EXCEEDED

You've exceeded the number of licenses assigned to your org.

LIGHT_PORTAL_USER_EXCEPTION

You attempted an action with a customer portal that's not allowed. For example, trying to add the user to a case team.

LIMIT_EXCEEDED

You've exceeded a limit on a field size or value, license, platform event publishing, or other component.

LOGIN_CHALLENGE_ISSUED

An email containing a security token was sent to the user's email address because the user logged in from an untrusted IP address. The user can't log in until the security token is added to the end of the password.

LOGIN_CHALLENGE_PENDING

The user logged in from an untrusted IP address, but a security token hasn't yet been issued.

LOGIN_MUST_USE_SECURITY_TOKEN

The user must add a security token to the end of the password to log in.

MALFORMED_ID

An ID must be either 15 characters, or 18 characters with a valid case-insensitive extension. There's also an exception code of the same name.

MANAGER_NOT_DEFINED

A manager hasn't been defined for the specified approval process.

MASSMAIL_RETRY_LIMIT_EXCEEDED

A mass mail retry failed because your org has exceeded its mass mail retry limit.

MASS_MAIL_LIMIT_EXCEEDED

The org has exceeded its daily limit for mass email. Mass email messages can't be sent again until the next day.

MAXIMUM_CCEMAILS_EXCEEDED

You've exceeded the maximum number of specified CC addresses in a workflow email alert.

MAXIMUM_DASHBOARD_COMPONENTS_EXCEEDED

You've exceeded the document size limit for a dashboard.

MAXIMUM_HIERARCHY_LEVELS_REACHED

You've reached the maximum number of levels in a hierarchy.

MAXIMUM_SIZE_OF_ATTACHMENT

You've exceeded the maximum size of an attachment.

MAXIMUM_SIZE_OF_DOCUMENT

You've exceeded the maximum size of a document.

MAX_ACTIONS_PER_RULE_EXCEEDED

You've exceeded the maximum number of actions per rule.

MAX_ACTIVE_RULES_EXCEEDED

You've exceeded the maximum number of active rules.

MAX_APPROVAL_STEPS_EXCEEDED

You've exceeded the maximum number of approval steps for an approval process.

MAX_FORMULAS_PER_RULE_EXCEEDED

You've exceeded the maximum number of formulas per rule.

MAX_RULES_EXCEEDED

You've exceeded the maximum number of rules for an object.

MAX_RULE_ENTRIES_EXCEEDED

You've exceeded the maximum number of entries for a rule.

MAX_TASK_DESCRIPTION_EXCEEDED

The task description is too long.

MAX_TM_RULES_EXCEEDED

You've exceeded the maximum number of rules per Territory.

MAX_TM_RULE_ITEMS_EXCEEDED

You've exceeded the maximum number of rule criteria per rule for a Territory.

MERGE_FAILED

A merge operation failed.

MISSING_ARGUMENT

You didn't specify a required argument.

NONUNIQUE_SHIPPING_ADDRESS

You can't insert a reduction order item if the original order shipping address is different from the shipping address of other items in the reduction order.

NO_APPLICABLE_PROCESS

A `process()` request failed because the record submitted doesn't satisfy the entry criteria of any active approval processes for which the user has permission.

NO_ATTACHMENT_PERMISSION

Your org doesn't permit email attachments.

NO_INACTIVE_DIVISION_MEMBERS

You can't add members to an inactive Division.

NO_MASS_MAIL_PERMISSION

You don't have permission to send the email. You must have "Mass Email" to send mass mail or "Send Email" to send individual email.

NUMBER_OUTSIDE_VALID_RANGE

The number specified is outside the valid range of values.

NUM_HISTORY_FIELDS_BY_SUBJECT_EXCEEDED

The number of history fields specified for the sObject exceeds the allowed limit.

OP_WITH_INVALID_USER_TYPE_EXCEPTION

The operation you attempted can't be performed for one or more users. For example, you can't add high-volume portal users to a group.

OPTED_OUT_OF_MASS_MAIL

An email can't be sent because the specified User has opted out of mass mail.

ORDER_MANAGEMENT_ACTION_NOT_ALLOWED

The requested action isn't allowed.

ORDER_MANAGEMENT_RECORD_EXISTS

You can't create the record because it already exists.

ORDER_MANAGEMENT_RECORD_NOT_FOUND

We couldn't find the requested record.

PACKAGE_LICENSE_REQUIRED

The logged-in user can't access an object that is in a licensed package without a license for the package.

PLATFORM_EVENT_ENCRYPTION_ERROR

The platform event messages couldn't be published due to a problem with encryption. A misconfiguration in your Salesforce org or a general encryption service error can cause this problem.

PLATFORM_EVENT_PUBLISHING_UNAVAILABLE

Publishing platform event messages failed due to a service being temporarily unavailable. Try again later.

PLATFORM_EVENT_PUBLISH_FAILED

The platform event message couldn't be published after one or more attempts because of a system error. Try again later.

PORTAL_USER_ALREADY_EXISTS_FOR_CONTACT

A create [User](#) operation failed because you can't create a second portal user under a Contact.

PRIVATE_CONTACT_ON_ASSET

You can't have a private contact on an asset.

RECORD_IN_USE_BY_WORKFLOW

You can't access a record that's in use by a workflow or approval process.

REQUEST_RUNNING_TOO_LONG

A request that has been running too long is canceled.

REQUIRED_FIELD_MISSING

A call requires a field that wasn't specified.

SELF_REFERENCE_FROM_TRIGGER

You can't recursively update or delete the same object from an Apex trigger. This error often occurs when:

- You try to update or delete an object from within its before trigger.
- You try to delete an object from within its after trigger.

This error occurs with both direct and indirect operations. The following is an example of an indirect operation:

1. A request is submitted to update Object A.
2. A `before update` trigger on object A creates an object B.
3. Object A is updated.
4. An `after insert` trigger on object B queries object A and updates it. This update is an indirect update of object A because of the before trigger of object A, so an error is generated.

SHARE_NEEDED_FOR_CHILD_OWNER

If a parent record has a child record that needs a sharing rule, you can't delete the sharing rule for the parent record.

SINGLE_EMAIL_LIMIT_EXCEEDED

(API version 18.0 and later) The org has exceeded its daily limit for individual emails. Individual email messages can't be sent again until the next day.

STANDARD_PRICE_NOT_DEFINED

Custom prices can't be defined without corresponding standard prices.

STORAGE_LIMIT_EXCEEDED

You've exceeded your org's storage limit.

STRING_TOO_LONG

The specified string exceeds the maximum allowed length.

TABSET_LIMIT_EXCEEDED

You've exceeded the number of tabs allowed for a tabset.

TEMPLATE_NOT_ACTIVE

The template specified is unavailable. Specify another template or make the template available for use.

TERRITORY_REALIGN_IN_PROGRESS

An operation can't be performed because a territory realignment is in progress.

TEXT_DATA_OUTSIDE_SUPPORTED_CHARSET

The specified text uses a character set that isn't supported.

TOO_MANY_APEX_REQUESTS

Too many Apex requests have been sent. This error is transient. Resend your request after a short wait.

TOO_MANY_ENUM_VALUE

A request failed because too many values were passed in for a multi-select picklist. You can select a maximum of 100 values for a multi-select picklist.

TRANSFER_REQUIRES_READ

You can't assign the record to the specified User because the user doesn't have read permission.

UNABLE_TO_LOCK_ROW

A deadlock or timeout condition has been detected.

- A deadlock involves at least two transactions that are attempting to update overlapping sets of objects. If the transaction involves a summary field, the parent objects are locked, making these transactions especially prone to deadlocks. To debug, check your code for deadlocks and correct. Deadlocks are generally not the result of an issue with Salesforce operations.
- A timeout occurs when a transaction takes too long to complete, for example, when replacing a value in a picklist or changing a custom field definition. The timeout state is temporary. No corrective action is needed.

If an object in a batch can't be locked, the entire batch fails with this error. Errors with this status code contain the IDs of the records that couldn't be locked, when available, in the error message.

UNAVAILABLE_RECORDTYPE_EXCEPTION

The appropriate default record type couldn't be found.

UNDELETE_FAILED

An object couldn't not be undeleted because it doesn't exist or hasn't been deleted.

UNKNOWN_EXCEPTION

The system encountered an internal error. Report this problem to Salesforce.



Note: Don't report this exception code to Salesforce if it results from a `sendEmail()` call. The `sendEmail()` call returns this exception code when it is used to send an email to one or more recipients who have the **Email Opt Out** option selected.

UNSPECIFIED_EMAIL_ADDRESS

The specified user doesn't have an email address.

UNSUPPORTED_APEX_TRIGGER_OPERATION

You can't save recurring events with an Apex trigger.

UNVERIFIED_SENDER_ADDRESS

A `sendEmail()` call attempted to use an unverified email address defined in the `OrgWideEmailAddress` object.

WEBLINK_SIZE_LIMIT_EXCEEDED

The size of a WebLink URL or JavaScript code exceeds the limit.

WEBLINK_URL_INVALID

The WebLink URL has failed the URL string validation check.

WRONG_CONTROLLER_TYPE

The controller type for your Visualforce email template doesn't match the object type being used.

If you receive a status code not listed in the previous table, contact Customer Support.

ExtendedErrorDetails

An `ExtendedErrorDetails` element contains additional information about an error. The `ExtendedErrorDetails` element can include the following properties.

Name	Type	Description
<code>extendedErrorCode</code>	<code>ExtendedErrorCode</code>	A code that characterizes the extended error details.
<i>extended error property</i>	Varies	An extended error property that contains more information about the error. The property name and value vary based on the extended error code. For example, in a limits-related error, the property <code>limit</code> contains the value of a limit and <code>input</code> contains the submitted value that reaches or exceeds the <code>limit</code> .

ExtendedErrorCode

Each `ExtendedErrorDetails` contains an error code and its related properties. The codes and descriptions included here are examples of extended errors. You might see other extended error codes, depending on the error you encounter.

MAX_STATEMENT_SIZE

The query exceeds the character limit. See [SOQL SELECT Syntax](#).

MAX_XDS_IMPLICIT_SUBQUERIES

The query exceeds the number of joins allowed across external objects. [Understanding Relationship Query Limitations](#).

Duplicate Management Data Types

DuplicateError

Contains information about an error that occurred when an attempt was made to save a duplicate record. Use if your organization has set up duplicate rules, which are part of the Duplicate Management feature.

Fields

Field	Details
<code>duplicateResult</code>	<p>Type DuplicateResult</p> <p>Description The details of a duplicate rule and duplicate records found by the duplicate rule.</p>

Field	Details
fields	<p>Type string[]</p> <p>Description Array of one or more field names. Identifies which fields in the object, if any, affected the error condition.</p>
message	<p>Type string</p> <p>Description Error message text.</p>
statusCode	<p>Type StatusCode</p> <p>Description A code that characterizes the error. The full list of status codes is available in the WSDL file for your organization (see Generating the WSDL File for Your Organization).</p>

Usage

DuplicateError and its constituent objects are available to organizations that use duplicate rules.

DuplicateError is a data type of Error.

To process duplicates, loop through all the Error objects in the `errors` field on SaveResult. An Error object with a data type of DuplicateError contains information about an error that occurred when an attempt was made to save a duplicate record. To access information about the duplicates, use the `duplicateResult` field.

Java Sample

Here is a sample that shows how to see if there are any errors on the saveResult with a data type of DuplicateError. If so, duplicates were detected. See [DuplicateResult](#) for a full code sample that shows how to block users from entering duplicate leads and display an alert and a list of duplicates.

```

if (!saveResult.isSuccess()) {
    for (Error e : saveResult.getErrors()) {
        if (e instanceof DuplicateError) {
            System.out.println("Duplicate(s) Detected for lead with ID: " +
leads[i].getId());
            System.out.println("ERROR MESSAGE: " + e.getMessage());
            System.out.println("STATUS CODE: " + e.getStatusCode());
            DuplicateResult dupeResult = ((DuplicateError)e).getDuplicateResult();
            System.out.println("Found the following duplicates...");
            for (MatchResult m : dupeResult.getMatchResults()) {
                if (m.isSuccess()) {
                    System.out.println("The match rule that was triggered was " +
m.getRule());
                    for (MatchRecord mr : m.getMatchRecords()) {

```

```

    System.out.println("Your record matched " + mr.getRecord().getId()
+ " of type "
    + mr.getRecord().getType());
    System.out.println("The match confidence is " +
mr.getMatchConfidence());
    }
    }
    }
    }
    }
    }
    }
}

```

DuplicateResult

Represents the details of a duplicate rule that detected duplicate records and information about those duplicate records.

Fields

Field	Details
<code>allowSave</code>	<p>Type boolean</p> <p>Description true if duplicates are allowed to be saved. false if duplicates are not allowed to be saved.</p>
<code>duplicateRule</code>	<p>Type string</p> <p>Description The name of the duplicate rule that detected duplicate records.</p>
<code>duplicateRuleEntityType</code>	<p>Type string</p> <p>Description The name of the duplicate rule that detected duplicate records.</p>
<code>errorMessage</code>	<p>Type string</p> <p>Description The error message configured by the administrator to warn users they are potentially creating duplicate records. This message is associated with a duplicate rule.</p>
<code>matchResults</code>	<p>Type MatchResult</p> <p>Description The duplicate records and related match information.</p>

Usage

DuplicateResult and its constituent objects are available to organizations that use duplicate rules.

Java Sample

Here is a sample that shows how to block users from entering duplicate leads and display an alert and a list of duplicates.

```
package com.doc.example;

import java.io.FileNotFoundException;

import com.sforce.soap.partner.*;
import com.sforce.soap.partner.Error;
import com.sforce.soap.partner.object.SObject;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;

public class SaveResultsWithDupeHeader {

    private PartnerConnection partnerConnection = null;
    static SaveResultsWithDupeHeader tester;

    public static void main(String[] args) {
        tester = new SaveResultsWithDupeHeader();
        try {
            tester.demoDuplicateRuleHeader();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /*
     * Make sure that you have an active lead duplicate rule linked to an active matching
     * rule. This method tries to save
     * an array of leads and inspects whether any duplicates were detected
     */
    public void demoDuplicateRuleHeader() throws FileNotFoundException, ConnectionException
    {
        // Create the configuration for the partner connection
        ConnectorConfig config = new ConnectorConfig();
        config.setUsername("user@domain.com");
        config.setPassword("secret");
        config.setAuthEndpoint("authEndPoint");
        config.setTraceFile("traceLogs.txt");
        config.setTraceMessage(true);
        config.setPrettyPrintXml(true);

        // Initialize the connection
        partnerConnection = new PartnerConnection(config);

        // Get the leads that have to be saved
        SObject[] leads = getLeadsForInsertOrUpdate();
    }
}
```

```

    /* Set a duplicate rule header to return a result if duplicates are detected
    * This sets the allowSave, includeRecordDetails, and runAsCurrentUser flags to
true
    */
    partnerConnection.setDuplicateRuleHeader(true, true, true);

    // Save the leads
    SaveResult[] saveResults = partnerConnection.create(leads);

    // Check the results to see if duplicates were detected
    for (int i = 0; i < leads.length; i++) {
        SaveResult saveResult = saveResults[i];
        if (!saveResult.isSuccess()) {
            for (Error e : saveResult.getErrors()) {
                // See if there are any errors on the saveResult with a data type of
DuplicateError
                if (e instanceof DuplicateError) {
                    System.out.println("Duplicate(s) Detected for lead with ID: " +
leads[i].getId());

                    System.out.println("ERROR MESSAGE: " + e.getMessage());
                    System.out.println("STATUS CODE: " + e.getStatusCode());
                    DuplicateResult dupeResult =
((DuplicateError)e).getDuplicateResult();
                    System.out.println("Found the following duplicates...");
                    for (MatchResult m : dupeResult.getMatchResults()) {
                        if (m.isSuccess()) {
                            System.out.println("The match rule that was triggered was
" + m.getRule());

                                for (MatchRecord mr : m.getMatchRecords()) {
                                    System.out.println("Your record matched " +
mr.getRecord().getId() + " of type "
                                        + mr.getRecord().getType());
                                    System.out.println("The match confidence is " +
mr.getMatchConfidence());

                                        for (FieldDiff f : mr.getFieldDiffs()) {
                                            System.out.println("For field " + f.getName() + "
field difference is "
                                                + f.getDifference().name());
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        // Clear the duplicate rule header
        partnerConnection.clearDuplicateRuleHeader();
    }

    /**
    * The assumption here is that this method is retrieving leads from either UI, a data
source, etc

```

```

    */
    private SObject[] getLeadsForInsertOrUpdate() {
        return new SObject[0];
    }
}

```

MatchResult

Represents the duplicate results for a matching rule.

Fields

Field	Details
errors	<p>Type Error[]</p> <p>Description Errors that occurred during matching for the matching rule.</p>
entityType	<p>Type string</p> <p>Description The entity type of the matching rule.</p>
matchEngine	<p>Type string</p> <p>Description The match engine for the matching rule.</p>
matchRecords	<p>Type MatchRecord[]</p> <p>Description Information about the duplicates detected by the matching rule.</p>
rule	<p>Type string</p> <p>Description The developer name of the matching rule that detected duplicates.</p>
size	<p>Type int</p> <p>Description The number of duplicates detected by the matching rule.</p>

Field	Details
success	<p>Type boolean</p> <p>Description true if the matching rule successfully ran. false if there's an error with the matching rule.</p>

Usage

MatchResult and its constituent objects are available to organizations that use duplicate rules.

There is 1 MatchResult for each saved record that has duplicates. The MatchResult contains all duplicates for that saved record.

Java Sample

Here is a sample that shows how to display the ID and type of all duplicates detected when leads are saved. See [DuplicateResult](#) for a full code sample that shows how to block users from entering duplicate leads and display an alert and a list of duplicates.

```
for (MatchResult m : dupeResult.getMatchResults()) {
    if (m.isSuccess()) {
        System.out.println("The match rule that was triggered was " + m.getRule());
        for (MatchRecord mr : m.getMatchRecords()) {
            System.out.println("Your record matched " + mr.getRecord().getId() + " of type "
                + mr.getRecord().getType());
            System.out.println("The match confidence is " + mr.getMatchConfidence());
        }
    }
}
```

MatchRecord

Represents a duplicate record detected by a matching rule.

Fields

Field	Details
additionalInformation	<p>Type AdditionalInformationMap</p> <p>Description Other information about matched records.</p>
fieldDiffs	<p>Type FieldDiff[]</p>

Field	Details
	<p>Description</p> <p>Matching rule fields and how each field value compares for the duplicate and its matching record.</p>
matchConfidence	<p>Type</p> <p>double</p> <p>Description</p> <p>The ranking of how similar a matched record's data is to the data in your request. Must be equal to or greater than the value of the <code>minMatchConfidence</code> specified in your request. Returns <code>-1</code> if unused.</p>
record	<p>Type</p> <p>sObject</p> <p>Description</p> <p>The fields and field values for the duplicate record.</p>

Usage

MatchRecord and its constituent objects are available to organizations that use duplicate rules.

Each MatchRecord represents a duplicate detected when a record is saved. There can be multiple MatchRecord objects for a saved record if multiple duplicates are detected.

Java Sample

Here is a sample that shows how to display the ID and type of all duplicates detected when a lead is saved. See [DuplicateResult](#) for a full code sample that shows how to block users from entering duplicate leads and display an alert and a list of duplicates.

```
for (MatchRecord mr : m.getMatchRecords()) {
    System.out.println("Your record matched " + mr.getRecord().getId() + " of type "
        + mr.getRecord().getType());
    System.out.println("The match confidence is " + mr.getMatchConfidence());
}
```

FieldDiff

Represents the name of a matching rule field and how the values of the field compare for the duplicate and its matching record.

Fields

Field	Details
difference	<p>Type</p> <p>differenceType</p>

Field	Details
	<p>Description</p> <p>How the values of the matching rule field compare for the duplicate and its matching record.</p> <p>Possible values include:</p> <ul style="list-style-type: none"> • Same: Indicates the field values match exactly. • Different: Indicates that the field values do not match. • Null: Indicates that the field values are a match because both values are blank.
name	<p>Type</p> <p>string</p> <p>Description</p> <p>The name of a field on a matching rule that detected duplicates.</p>

Java Sample

Here is a sample that shows how to display the matching rule field differences when a duplicate is detected. See [DuplicateResult](#) for a full code sample that shows how to block users from entering duplicate leads and display an alert and a list of duplicates.

```
for (FieldDiff f : mr.getFieldDiffs()) {
    System.out.println("For field " + f.getName() + " field difference is "
        + f.getDifference().name());
}
```

AdditionalInformationMap

Represents other information, if any, about matched records.

Fields

Field	Details
name	<p>Type</p> <p>string</p> <p>Description</p> <p>The name of the element.</p>
value	<p>Type</p> <p>string</p> <p>Description</p> <p>The value of the element.</p>

CHAPTER 3 Tooling API Objects in the Enterprise WSDL

Some objects used by the Tooling API are included in the Enterprise and Partner WSDL. Use the objects from these WSDLs instead of the Tooling WSDL if you need field-level security.

The following Tooling API objects are exposed in the Enterprise and Partner WSDL.

- BriefcaseDefinition
- DataType
- EntityDefinition
- EntityParticle
- FieldDefinition
- PicklistValueInfo
- Publisher
- SearchLayout
- Service
- ServiceDataType (Reserved for future use.)
- ServiceFieldDataType (Unavailable in version 35.0 and later. Do not use.)
- RelationshipDomain
- RelationshipInfo
- UserEntityAccess
- UserFieldAccess
- WSDLDataType (Reserved for future use.)
- XmlSchema (Reserved for future use.)

For more information, use the *Tooling API Developer's Guide*.

CHAPTER 4 API Call Basics

In this chapter ...

- [Characteristics of API Calls](#)
- [Factors that Affect Data Access](#)
- [Package Version Settings](#)

API calls represent specific operations that your client applications can invoke at runtime to perform tasks, for example:

- Query data in your organization.
- Add, update, and delete data.
- Obtain metadata about your data.
- Run utilities to perform administration tasks.

Using your development environment, you can construct Web service client applications that use standard Web service protocols to programmatically:

- Log in to the login server (`login()`) and receive authentication information to be used for subsequent calls
- Query your organization's information (`query()`, `queryAll()`, `queryMore()`, and `retrieve()` calls)
- Perform text searches across your organization's information (`search()` call)
- Create, update, and delete data (`create()`, `merge()`, `update()`, `upsert()`, `delete()`, and `undelete()` calls)
- Perform administrative tasks, such as retrieving user information (`getUserInfo()` call), changing passwords (`setPassword()` and `resetPassword()` calls), and getting the system time (`getServerTimestamp()` call)
- Replicate data locally (`getDeleted()` and `getUpdated()` calls)
- Obtain and navigate metadata about your organization's data (`describeGlobal()`, `describeSubject()`, `describeSubjects()`, `describeLayout()`, and `describeTabs()` calls)
- Work with approval processes (`process()`)
- Return category groups and categories from your organization (`describeDataCategoryGroups()` and `describeDataCategoryGroupStructures()`).

See [Core Calls](#), [Describe Calls](#), and [Utility Calls](#) for complete details about each call.

SEE ALSO:

[User Permissions](#)

Characteristics of API Calls

All API calls are:

- **Service Requests and Responses**—Your client application prepares and submits a service request to the Lightning Platform Web Service via the API, the Lightning Platform Web Service processes the request and returns a response, and the client application handles the response.
- **Synchronous**—After the API call is invoked, your client application waits until it receives a response from the service. Asynchronous calls are not supported.
- **Committed Automatically Versus Rollback on Error**—By default, every operation that writes to a Salesforce object is committed automatically. This is analogous to the AUTOCOMMIT setting in SQL. For `create()`, `update()`, and `delete()` calls that attempt to write to multiple records for an object, the write operation for each record is treated as a separate transaction. For example, if a client application attempts to create two new accounts, they're created using mutually exclusive insert operations that succeed or fail individually, not as a group.

For API version 20.0 and later, there is an `AllOrNoneHeader` header that allows a call to roll back all changes unless all records are processed successfully. This header is supported by the `create()`, `delete()`, `undelete()`, `update()`, and `upsert()` calls.

- **Note:** The default behavior means that client applications may need to handle some failures: for example, if you create an opportunity that has shipments (a custom object), and the opportunity line item gets created but the shipment creation fails, if your business rules required all opportunities be created with shipment, your client application would need to roll back the opportunity creation. The easiest way to do this is to use `AllOrNoneHeader`.

Factors that Affect Data Access

When using the API, the following factors affect access to your organization's data:

Access

Your organization must be enabled for API access.

Objects may not be available until you contact Salesforce and request access. For example, Territory2 is visible only if Enterprise Territory Management has been enabled in the application. Such requirements are in the "Usage" section for each object.

Sometimes a feature must be used once before objects related to it can be accessed with the API. For example, the `recordTypeId`s is available only after at least one record type has been created for your organization in the user interface.

To investigate data access issues, you can start by inspecting the WSDL:

- **Enterprise WSDL:** The generated enterprise WSDL file contains all of the objects that are available to your organization. By using the API, a client application can access objects that are defined in your enterprise WSDL file.
- **Partner WSDL:** When using the generated partner WSDL file, a client application can access objects that are returned in the `describeGlobal()` call.

Object-Level and Field-Level Security

The API respects object-level and field-level security configured in the user interface. You can access objects and fields only if the logged-in user's permissions and access settings allow such access. For example, fields that are not visible to a given user are not returned in a `query()` or `describeObjects()` call. Similarly, read-only fields can't be updated.

User Permissions

A user attempting to access the API must have the permission "API Enabled" selected. It's selected by default.

Your client application logs in as a user called a *logged-in* user. The logged-in user's permissions grant or deny access to specific objects and fields in your organization:

- **Read**—Users can only view objects of this type.
- **Create**—Users can read and create objects of this type.
- **Edit**—Users can read and update objects of this type.
- **Delete**—Users can read, edit, and delete objects of this type.

User permissions do not affect field-level security. If field-level security specifies that a field is hidden, users with “Read” on that object can view only those fields that are not hidden on the record. In addition, users with “Read” on an object can view only those records that sharing settings allow. The one exception is the “Edit Read Only Fields” permission, which gives users the ability to edit fields marked as read only via field-level security.

Sharing

For most API calls, data that is outside of the logged-in user’s sharing model is not returned. Users are granted the most permissive access that is available to them, either through organization-wide defaults or manual record sharing, just as in the application.

User Permissions that Override Sharing

- **View All**—Users can view all records associated with this object, regardless of sharing settings.
- **Modify All**—Users can read, edit, delete, transfer, and approve all records associated with this object, regardless of sharing settings.
- **Modify All Data**—users can read, edit, delete, transfer, and approve all records regardless of sharing settings. This permission is not an object-level permission, unlike “View All” and “Modify All.”

To protect the security of your data, give the logged-in user only the permissions needed to successfully execute all the calls made by the application. For large integration applications, “Modify All Data” may speed up call response times. If you are loading a large number of records, use [Bulk API 2.0](#) instead.

Related Objects

Some objects depend on other objects for permission. For example, AccountTeamMember follows sharing on the associated permission-assigned object such as the Account record. Similarly, a Partner depends on the permissions in the associated .

Ownership changes to a record do not automatically cascade to related records. For example, if ownership changes for a given Account, ownership does not then automatically change for any Contract associated with that Account—each ownership change must be made separately and explicitly by the client application.

Object Properties

To create an object with the `create()` call, the object’s `createable` attribute must be set to `true`. To determine what operations are allowed on a given object, your client application can invoke the `describeSObjects()` call on the object and inspect the properties in the `DescribeSObjectResult`.

 **Note:** `replicable` allows `getUpdated()` and `getDeleted()` calls.

Page Layouts and Record Types

Requirements defined in the Salesforce user interface for page layouts and record types are not enforced by the API:

- Page layouts can specify whether a given field is required, but the API does not enforce such layout-specific field restrictions or validations in `create()` and `update()` calls. It’s up to the client application to enforce any such constraints, if applicable.
- Record types can control which picklist values can be chosen in a given record and which page layouts users with different profiles can see. However, such rules that are configured and enforced in the user interface are not enforced in the API. For example, the API does not validate whether the value in a picklist field is allowed per any record type restrictions associated with the profile of the logged-in user. Similarly, the API does not prevent a client application from adding data to a particular field simply because that field does not appear in a layout associated with the profile of the logged-in user.

Referential Integrity

To ensure referential integrity, the API forces or prevents certain behaviors:

- ID values in [reference fields](#) are validated in `create()` and `update()` calls.

- If a client application deletes a record, then its children are automatically deleted as part of the call if the `cascadeDelete` property on `ChildRelationship` for that child has a value of `true`. For example, if a client application deletes an `Opportunity`, then any associated `OpportunityLineItem` records are also deleted. However, if an `OpportunityLineItem` is not deletable or is currently being used, then deletion of the parent `Opportunity` fails. For example, if a client application deletes an `Invoice_Statement`, then any associated `Line_Item` records are also deleted. However, if a `Line_Item` is not deletable or is currently being used, then deletion of the parent `Invoice_Statement` fails. Use `DescribeObjectResult` to view the `ChildRelationship` value if you want to be sure what will be deleted.

There are certain exceptions that prevent the execution of a `cascadeDelete`. For example, you can't delete an account if it has associated cases, if it has related opportunities that are owned by other users, or if associated contacts are enabled for the Customer Portal. In addition, if you attempt to delete an account that has closed/won opportunities owned by you or has active contracts, then the delete request for that record will fail.

Package Version Settings

When your API client is referencing components in managed packages, you can specify the version of each installed package that you want to reference for your integration. This allows your API client to continue to function with specific, known behavior even when you install subsequent versions of a package. You can use the `PackageVersionHeader` SOAP header to set different package versions for different calls, if necessary.

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. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package.

Default package versions for API calls provide fallback settings if package versions are not provided by an API call. Many API clients do not include package version information, so the default settings maintain existing behavior for these clients.

You can specify the default package versions for enterprise API and partner API calls. The enterprise WSDL is for customers who want to build an integration with their Salesforce organization only. It is strongly typed, which means that calls operate on objects and fields with specific data types, such as `int` and `string`. The partner WSDL is for customers, partners, and ISVs who want to build an integration that can work across multiple Salesforce organizations, regardless of their custom objects or fields. It is loosely typed, which means that calls operate on name-value pairs of field names and values instead of specific data types.

You must associate the enterprise WSDL with specific package versions to maintain existing behavior for clients. There are options for setting the package version bindings for an API call from client applications using either the enterprise or partner WSDL. The package version information for API calls issued from a client application based on the enterprise WSDL is determined by the first match in the following settings.

1. The `PackageVersionHeader` SOAP header.
2. The SOAP endpoint contains a URL with a format of `serverName/services/Soap/c/api_version/ID` where `api_version` is the version of the API, such as 55.0, and `ID` encodes your package version selections when the enterprise WSDL was generated.
3. The default enterprise package version settings.

The partner WSDL is more flexible as it is used for integration with multiple organizations. If you choose the Not Specified option for a package version when configuring the default partner package versions, the behavior is defined by the latest installed package version. This means that behavior of package components, such as an Apex trigger, could change when a package is upgraded and that change would immediately impact the integration. Subscribers may want to select a specific version for an installed package for all partner API calls from client applications to ensure that subsequent installations of package versions do not affect their existing integrations.

The package version information for partner API calls is determined by the first match in the following settings.

1. The PackageVersionHeader SOAP header.
2. An API call from a Visualforce page uses the package versions set for the Visualforce page.
3. The default partner package version settings.

To configure default package versions for API calls:

1. From Setup, enter *API* in the *Quick Find* box, then select **API**.
2. Click **Configure Enterprise Package Version Settings** or **Configure Partner Package Version Settings**. These links are only available if you have at least one managed package installed in your organization.
3. Select a *Package Version* for each of your installed managed packages. If you are unsure which package version to select, you should leave the default selection.
4. Click **Save**.



Note: Installing a new version of a package in your organization does not affect the current default settings.

CHAPTER 5 Error Handling

In this chapter ...

- [Error Handling for Session Expiration](#)
- [More About Error Handling](#)

The API calls return error data that your client application can use to identify and resolve runtime errors. If an error occurs during the invocation of most API calls, then the API provides the following types of error handling:

- For errors resulting from badly formed messages, failed authentication, or similar problems, the API returns a SOAP fault message with an associated [ExceptionCode](#).
- For most calls, if the error occurs because of a problem specific to the query, the API returns an [Error](#). For example, if a `create()` request contains more than 200 objects.

Error Handling for Session Expiration

When you sign on via the `login()` call, a new client session begins and a corresponding unique session ID is generated. Sessions expire automatically after a predetermined length of inactivity, which can be configured in Salesforce from Setup by clicking **Security Controls**. The default is 120 minutes (two hours). If you make an API call, the inactivity timer is reset to zero.

When your session expires, the exception code `INVALID_SESSION_ID` is returned. If this happens, you must invoke the `login()` call again.

More About Error Handling

For more information about errors, see the following topics:

- [API Fault Element](#)
- [ExceptionCode](#)
- [Error](#)

CHAPTER 6 Security and the API

In this chapter ...

- [User Authentication](#)
- [User Profile and Permission Sets Configuration](#)
- [Security Token](#)
- [Sharing](#)
- [Implicit Restrictions for Objects and Fields](#)
- [API Access in Salesforce AppExchange Packages](#)
- [Outbound Port Restrictions](#)

Client apps that access your Salesforce data are subject to the same security protections that are used in the Salesforce user interface. Additional protection is available for orgs that install AppExchange managed packages if those packages contain components that access Salesforce via the API.

User Authentication

Client apps must log in using valid credentials. After validating, the server provides the client app with a:


- `sessionId` that is set into the session header so that all subsequent calls to the Web service are authenticated
- URL address (`serverUrl`) for the client app's web service requests

Salesforce supports only the Transport Layer Security (TLS) protocol and `frontdoor.jsp`. Ciphers must have a key length of at least 128 bits.

User Profile and Permission Sets Configuration

As an org's Salesforce admin, you control which features and views are available to users by configuring profiles and permission sets and assigning users to them. To access the API to issue calls and receive the call results, a user must have the API Enabled permission. Client apps can query or update only those objects and fields to which they have access via the permissions of the logged-in user.

To create, edit, or delete a profile, from Setup, enter *Profiles* in the Quick Find box, then select **Profiles**. To create, edit, or delete a permission set, from Setup, enter *Permission Sets* in the Quick Find box, then select **Permission Sets**.

 **Note:** The web services WSDL files return all available objects and fields for an org.


Security Token

When users log in to Salesforce via the user interface, the API, or a desktop client such as Salesforce for Outlook, Connect Offline, Connect for Office, or the Data Loader, Salesforce authorizes the login as follows.

1. Salesforce checks whether the user's profile has login-hour restrictions. If the user's profile specifies login-hour restrictions, login attempts outside the specified hours are denied.
2. If the user has the Multi-Factor Authentication for User Interface Logins permission, the Salesforce login process prompts the user for an identity verification method in addition to their username and password. If the user's account isn't already connected to a verification method, such as the Salesforce Authenticator mobile app, Salesforce prompts the user to register a method.
3. If the user has the Multi-Factor Authentication for API Logins permission and connected an authenticator app to the account, the user must enter a verification code (TOTP) generated by the authenticator app. If the user uses the standard security token, Salesforce returns an error.
4. Salesforce then checks whether the user's profile defines IP address range restrictions. If so, logins from outside the IP address range are denied. If the **Enforce login IP ranges on every request** session setting is enabled, the IP address restrictions are enforced for each page request, including requests from client apps.
5. If profile-based IP address restrictions aren't set, Salesforce checks whether the user is logging in from a device that was previously used to access Salesforce.
 - If the user is logging in from a device and browser that Salesforce recognizes, the login is allowed.
 - If the user is logging in from an IP address on your org's trusted IP address list, the login is allowed.
 - If the user isn't logging in from a trusted IP address, device, or browser that Salesforce recognizes, the login is blocked.

Whenever a login is blocked or returns an API login fault, Salesforce verifies the user's identity.


- For access via the user interface, the user is prompted to verify using Salesforce Authenticator (version 2 or later) or enter a verification code.

 **Note:** Users aren't asked for a verification code the first time they log in to Salesforce.

- For access via the API or client app, if the Multi-Factor Authentication on API Logins permission is set on the user profile, users enter a TOTP verification code generated by an authenticator app.

If the permission isn't set, users must add their security token to the end of their password to log in. A security token is a generated key from Salesforce. For example, if a user's password is *mypassword* and the security token is *XXXXXXXXXX*, the user enters *mypasswordXXXXXXXXXX* to log in. Some client apps have a separate field for the security token.

Users can get their security token by changing their password or resetting their security token via the Salesforce user interface. When a user changes a password or resets a security token, Salesforce sends a new security token to the email address on the user's Salesforce record. The security token is valid until the user resets the security token, changes a password, or has a password reset.

 **Note:** Before you access Salesforce from a new IP address, we recommend that you get your security token from a trusted network using Reset My Security Token.

For more information about tokens, see [Reset Your Security Token](#) in Salesforce Help.

When a user's password is changed, the user's security token is automatically reset. To log in after the reset, the user adds the generated security token to the end of the password. Or the user enters the new password after you add the user's IP address to the org's list of trusted IP addresses.

If you reset the password of a user with the API only user permission, the user doesn't receive a security token reset email. If you want API only users to receive a security token reset email when you reset their password, take these steps.

- Temporarily assign the user to a profile that doesn't have the API only user permission. For more information on user profiles and permissions, see [User Permissions and Access](#).
- Ask the user to [manually reset their security token](#).
- Reassign the user to a profile with the API only user permission.

If single sign-on (SSO) is enabled, users who access the API or a desktop client can't log in unless their IP address is included on your org's list of trusted IP addresses or on their profile, if their profile has IP address restrictions set. The delegated authentication authority usually handles login lockout policies for users with the Uses Single Sign-On permission. However, if the security token is enabled, your login lockout settings determine how many times a user can try to log in with an invalid security token before getting locked out. For more information, see [Setting Login Restrictions and Setting Password Policies](#) in Salesforce Help.

Sharing

Sharing refers to the act of granting read or write access to a user or group so that they can view or edit a record owned by other users, if the default access levels don't permit such access. All API calls respect the sharing model.


The following table describes the types of access levels.

API Value	Salesforce User Interface Label	API Picklist Label	Description
None	Private	Private	Only the record owner and Users above that role in the hierarchy can view and edit the record.
Read	Read Only	Read Only	All Users and Groups can view the record but not edit it. Only the owner and users above that role in the hierarchy can edit the record.
Edit	Read/Write	Read/Write	All Users and Groups can view and edit the record.

API Value	Salesforce User Interface Label	API Picklist Label	Description
ReadEditTransfer	Read/Write/Transfer	Read/Write/Transfer	All Users and Groups can view, edit, delete, and transfer the record. (Only available for cases and leads as an org-wide default setting.)
All	Full Access	Owner	All Users and Groups can view, edit, transfer, delete, and share the record. (Only available for campaigns as an org-wide default setting.)
ControlledByParent	Controlled by Parent	Controlled By Parent	(Contacts only.) All Users and Groups can perform an action (such as view, edit, or delete) on the contact based on whether he or she can perform that same action on the record associated with it.

Not all access levels are available for every object. See the Fields table for each object to learn which access levels are available, as well as other sharing details specific to that object.

For more information about sharing in general, see [Salesforce Help](#).

 **Note:** In the API, you can create and update objects such as [AccountShare](#) and [OpportunityShare](#) that define sharing entries for records.

Implicit Restrictions for Objects and Fields

Certain objects can be created or deleted only in the Salesforce user interface. Other objects are read-only—client apps cannot create(), delete(), or update() such objects. Similarly, certain fields within some objects can be specified on create() but not on update(). Other fields are read-only—client apps cannot specify field values in create() or update() calls. For more information, see the respective object descriptions in [Object Basics](#).

API Access in Salesforce AppExchange Packages

The API allows access to objects and calls based on the permissions of the user who logs into the API. To prevent security issues from arising when installed packages have components that access data via the API, Salesforce provides additional security:

- When a developer creates an AppExchange package with components that access the API, the developer can restrict the API access for those components.
- When an administrator installs an AppExchange package, the administrator can accept or reject the access. Rejecting the access cancels the installation.
- After an administrator installs a package, the administrator can restrict the API access of components in the package that access the API.

Editing API access for a package is done in the Salesforce user interface. For more information, see *Manage API and Dynamic Apex Access in Packages* in [Salesforce Help](#).

API access for a package affects the API requests originating from components within the package; it determines the objects that the API requests can access. If the API access for a package is not defined, then the objects that the API requests have access to are determined by the user's permissions.

The API access for a package never allows users to do more than the permissions granted to the user. API access in a package only reduces what the user's permissions allow.

Choosing `Restricted` for the `API Access` setting in a package affects the following:

- API access in a package overrides the following user permissions:
 - Author Apex
 - Customize Application
 - Edit HTML Templates
 - Edit Read Only Fields
 - Manage Billing
 - Manage Call Centers
 - Manage Categories
 - Manage Custom Report Types
 - Manage Dashboards
 - Manage Letterheads
 - Manage Package Licenses
 - Manage Public Documents
 - Manage Public List Views
 - Manage Public Reports
 - Manage Public Templates
 - Manage Users
 - Transfer Record
 - Use Team Reassignment Wizards
 - View Setup and Configuration
 - Weekly Export Data
- If `Read`, `Create`, `Edit`, and `Delete` access are not selected in the API access setting for objects, users do not have access to those objects from the package components, even if the user has the “Modify All Data” and “View All Data” permissions.
- A package with `Restricted` API access can't create new users.
- Salesforce denies access to Web service and `executeanonymous` requests from an AppExchange package that has `Restricted` access.

The following considerations also apply to API access in packages:

- Workflow rules and Apex triggers fire regardless of API access in a package.
- If a component is in more than one package in an organization, API access is unrestricted for that component in all packages in the organization regardless of the access setting.
- If Salesforce introduces a new standard object after you select restricted access for a package, access to the new standard object is not granted by default. You must modify the restricted access setting to include the new standard object.
- When you upgrade a package, changes to the API access are ignored even if the developer specified them. This ensures that the administrator installing the upgrade has full control. Installers should carefully examine the changes in package access in each upgrade during installation and note all acceptable changes. Then, because those changes are ignored, the administrator should manually apply any acceptable changes after installing an upgrade.

- S-controls are served by Salesforce and rendered inline in Salesforce. Because of this tight integration, there are several means by which an s-control in an installed package could escalate its privileges to the user's full privileges. In order to protect the security of organizations that install packages, s-controls have the following limitations:
 - For packages you are developing (that is, not installed from AppExchange), you can only add s-controls to packages with the default `Unrestricted` API access. Once a package has an s-control, you cannot enable `Restricted` API access.
 - For packages you have installed, you can enable access restrictions even if the package contains s-controls. However, access restrictions provide only limited protection for s-controls. Salesforce recommends that you understand the JavaScript in an s-control before relying on access restriction for s-control security.
 - If an installed package has `Restricted` API access, upgrades will be successful only if the upgraded version does not contain any s-controls. If s-controls are present in the upgraded version, you must change the currently installed package to `Unrestricted` API access.

To manage API access to packages, see “Manage API and Dynamic Apex Access in Packages” in Salesforce Help.

 **Note:** XML-RPC requests that originate from restricted packages are denied access.

Outbound Port Restrictions

For security reasons, Salesforce restricts the outbound ports you can 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.

The port restriction applies to any feature where a port is specified, for example outbound messages, AJAX proxy, or single-sign on.

CHAPTER 7 Using the Partner WSDL

In this chapter ...

- [Obtaining the Partner WSDL File](#)
- [Calls and the Partner WSDL](#)
- [Objects, Fields, and Field Data and the Partner WSDL](#)
- [Queries and the Partner WSDL](#)
- [Namespaces in the Partner WSDL](#)
- [Package Versions and the Partner WSDL](#)
- [User Interface Themes](#)
- [Examples Using the Partner WSDL](#)

The API provides two WSDLs to choose from:

- **Enterprise Web Services WSDL**—Used by enterprise developers to build client applications for a single Salesforce organization. The enterprise WSDL is strongly typed, which means that it contains objects and fields with specific data types, such as `int` and `string`. Customers who use the enterprise WSDL document must download and re-consume it when changes are made to the custom objects or fields in their org or when they want to use a different version of the API. To access the current WSDL for your organization, log in to your Salesforce organization and from Setup, enter `API` in the `Quick Find` box. Then, on the API page, select **Generate Enterprise WSDL**.
- **Partner Web Services WSDL**—Used for client applications that are metadata-driven and dynamic in nature. It is particularly—but not exclusively—useful to Salesforce partners who are building client applications for multiple organizations. As a loosely typed representation of the Salesforce data model that works with name-value pairs of field names and values instead of specific data types, it can be used to access data within any organization. This WSDL is most appropriate for developers of clients that can issue a query call to get information about an object before the client acts on the object. The partner WSDL document needs to be downloaded and consumed only once per version of the API. To access the current WSDL for your organization, log in to your Salesforce organization and from Setup, enter `API` in the `Quick Find` box. Then, on the API page, select **Generate Partner WSDL**.

In general, the enterprise WSDL is more straightforward to use, while the partner WSDL is more flexible and dynamically adaptable to different organizations, allowing you to write a single application that can be used for multiple users and multiple organizations.

High Precision Versions

If you require higher precision than the regular WSDLs provide, ask your account team about the “High Precision API” feature. When this feature is enabled, the WSDLs that you download (both Enterprise and Partner) use higher precision data types. For example, this feature is useful if your organization uses complex numerical formulas that are prone to rounding errors.

 **Note:** This feature is a limited pilot and is not currently a generally available feature.

If you have been using the regular version of the WSDL and change to the high precision version, perform the following checks:

1. Download the new WSDL.
2. Regenerate the stub code. (See [Setting Up Your Java Developer Environment](#).)
3. Verify that the type of variables used to store numeric values in your code can accommodate the new types.

Obtaining the Partner WSDL File

To use the partner WSDL, download a copy of the file using either of the following methods:

- Obtain it from your organization's Salesforce administrator, or
- Generate from Setup in Salesforce (enter *API* in the **Quick Find** box, then select **API**) according to the instructions in [Step 2: Generate or Obtain the Web Service WSDL](#).

While the enterprise WSDL file needs to be regenerated whenever custom fields or custom objects are added to an organization's Salesforce information, the partner WSDL file remains the same regardless of underlying changes in the organization's Salesforce data.

Calls and the Partner WSDL

The partner WSDL file defines exactly the same API calls found in the enterprise WSDL file. A client application using the partner WSDL will likely use the following API calls to determine an organization's metadata:

Task / Call	Description
<code>describeGlobal()</code>	Retrieves a list of available objects for your organization's data.
<code>describeLayout()</code>	Retrieves metadata about page layouts for the specified object type.
<code>describeSObject()</code>	<code>describeSObject()</code> has been superseded by <code>describeSObjects()</code> .
<code>describeSObjects()</code>	Use to obtain metadata for a given object. You can first call to retrieve a list of all objects for your organization, then iterate through the list and use to obtain metadata about individual objects.
<code>describeTabs()</code>	In the user interface, users have access to standard apps (and may also have access to custom apps) as listed in the Lightning Platform app menu at the top of the page. Selecting a standard app or custom app in the user interface allows the user to switch between the listed apps at any time.

To explore an organization's metadata, a client application can:

1. Call `describeGlobal()` to obtain a list of available objects.
2. In the returned `DescribeGlobalResult` object, retrieve an array of `DescribeGlobalSObjectResult` objects by calling `subjects`.
3. Get the `sObject` type name for each returned `sObject` by calling `name` on the `DescribeGlobalSObjectResult` objects.
4. The `DescribeGlobalSObjectResult` object provides some metadata about the `sObject`, such as whether the `sObject` is createable or updateable. If you want to get more information about particular `sObjects`, like their fields and child relationships, call `describeSObjects()` by passing it an array of the `sObject` type names that you're interested in obtaining more information about.

sObject Reference Reuse

An `sObject` reference can't be reused within a single operation.

Use a different reference. For example, the following code snippet creates an account and contact with a custom field and an event using two different references:

```
SObject account = new com.sforce.soap.partner.sobject.wsc.SObject();
account.setType("Account");
```

```

account.setField("Name", "myAccount");
account.setField("XID1__c", "1");
SObject refAcc1 = new com.sforce.soap.partner.object.wsc.SObject();
refAcc1.setType("Account");
refAcc1.setField("XID1__c", "1");
SObject refAcc2 = new com.sforce.soap.partner.object.wsc.SObject();
refAcc2.setType("Account");
refAcc2.setField("XID1__c", "1");

SObject contact = new com.sforce.soap.partner.object.wsc.SObject();
contact.setType("Contact");
contact.setField("LastName", "LName");
contact.setField("XID2__c", "2");
contact.setField("Account", refAcc1);
SObject refCon = new com.sforce.soap.partner.object.wsc.SObject();
contact.setType("Contact");
contact.setField("XID2__c", "2");

SObject event = new com.sforce.soap.partner.object.wsc.SObject();
contact.setType("Event");
contact.setField("Subject", "myEvent");
contact.setField("ActivityDateTime", Calendar.getInstance());
contact.setField("DurationInMinutes", 60);
contact.setField("Who", refCon);
contact.setField("What", refAcc2);

client.create(new SObject[] { account, contact, event}); // exception thrown here

```

Any call that takes a parameter of the form `sObject[] sObjects` is subject to this limitation.

Objects, Fields, and Field Data and the Partner WSDL

The enterprise WSDL file defines all the specific objects (such as Account and Contact) in a Salesforce org. In contrast, the partner WSDL file defines a single, generic object (`sObject`) that represents all the objects. For a particular object, its type is defined in the `name` field in the returned `DescribeSObjectResult`.

With the partner WSDL, your client application code handles fields as arrays of name-value pairs that represent the field data. When referring to the name of an individual field, use the value in its `name` field of the `Field` type in the `DescribeSObjectResult`.

Languages vary in the way they handle name-value pairs and map typed values to the primitive XML data types defined in SOAP messages. With the enterprise WSDL, the mapping is handled implicitly. With the partner WSDL, however, you manually manage values and data types when building client applications. Specify the object type before you assign field values. When specifying the value of a particular field, use a value that is valid for the field (range, format, and data type). Make sure that you understand the mapping between data types in your programming language and XML primitive data types. See [SOAPType](#) for more information.

Queries and the Partner WSDL

When using the `query()` call with the partner WSDL, consider the following guidelines:

- The `queryString` parameter is case-insensitive. The API will accept field names in the `fieldList` using any combination of uppercase and lowercase letters. However, in the [QueryResult](#), the case of field names (both predefined and custom fields) will match exactly

the value in the `name` field of the `Field` type in the [DescribeSObjectResult](#). It is recommended that you use the proper case when specifying fields in the `fieldList`.

- For the partner WSDL, the ordering of fields in the [QueryResult](#) is determined by the field order in the `fieldList`, not the field order in the WSDL file.
- The `fieldList` cannot contain duplicate field names. For example:
 - Invalid (returns an error): `"SELECT Firstname, Lastname, Firstname FROM User"`
 - Valid: `"SELECT Firstname, Lastname FROM User"`
- The [QueryResult](#) always contains all of the fields specified in the `fieldList`, even if some of the fields contain no data (`null`). Although SOAP allows you to omit fields that contain no values in the result set, the API always returns an array containing all fields.
- If you use the partner WSDL, a query that includes ID will return the ID field twice in the SOAP XML response data. Similarly, a query that does not include ID will return a single null ID field in the SOAP XML response data. For example, a query for `SELECT ID, FirstName, LastName FROM Contact` might return a SOAP XML response with records like:

```
<records xsi:type="sf:sObject" xmlns="urn:partner.soap.sforce.com">
  <sf:type>Contact</sf:type>
  <sf:Id>0038000000Frj0BQRW</sf:Id>
  <sf:Id>0038000000Frj0BQRW</sf:Id>
  <sf:FirstName>John</sf:FirstName>
  <sf:LastName>Smith</sf:LastName>
</records>
```

This is expected behavior and something to be aware of if you are accessing the full SOAP XML response data and not using WSC to access the web service response.

Namespaces in the Partner WSDL

In XML, every tag has a defined namespace. In the `enterprise.wsdl`, namespaces are handled implicitly. When using API calls with the partner WSDL, however, you need to explicitly specify the correct namespaces for API calls, objects, and fields, and faults. This rule applies to predefined and custom objects and fields.

For	Namespace
API Calls	<code>urn:partner.soap.sforce.com</code>
sObjects	<code>urn:object.partner.soap.sforce.com</code>
Fields	<code>urn:object.partner.soap.sforce.com</code>
Faults	<code>urn:fault.partner.soap.sforce.com</code>

Package Versions and the Partner WSDL

The partner WSDL is loosely typed. This makes it more flexible for partners who want to integrate with multiple organizations. Default package versions for API calls provide fallback settings if package versions are not provided by an API call.

The behavior of a package in partner API calls is defined by the latest installed package version if the default value (`Not Specified`) is selected for the installed package. This means that behavior of package components, such as an Apex trigger, could change when a package is upgraded and that change would immediately impact the integration. Subscribers may want to select a specific version for

an installed package for all partner API calls from client applications to ensure that subsequent installations of package versions do not affect their existing integrations.

An API client developer should communicate with the administrator of the default partner package version settings if these are two different roles in your organization and the developer recommends changing the settings. Alternatively, an API client developer can set the package versions in the [PackageVersionHeader](#) SOAP header for the client.

A partner that is developing a package that references another package should always supply version information for the base package in their partner API calls. This ensures that the extension package is not affected by a component being deprecated in the base package.

The package version information for partner API calls is determined by the first match in the following settings.

1. The PackageVersionHeader SOAP header.
2. An API call from a Visualforce page uses the package versions set for the Visualforce page.
3. The default partner package version settings.

To configure default package versions for API calls with the partner WSDL, see [Package Version Settings](#).

User Interface Themes

Back in the Winter '06 release, Salesforce started supporting multiple user interface themes, allowing you to use different sets of icons and colors for the user interface. But these user interface themes do not apply when your org is using Lightning Experience.

Two user interface themes match the earlier iterations of Salesforce.

- Theme3—The “Salesforce Classic 2010 user interface theme.” This interface was previously referred to as “Salesforce” or “new user interface theme.” You might also be familiar with it as the Salesforce *Aloha* interface.
- Theme2—The “Salesforce Classic 2005 user interface theme.” This interface was previously referred to as “Salesforce Classic” or the “classic user interface theme.”

The `getUserInfo()` call returns a `getUserInfoResult` object, which includes the `userUiSkin` property. This property informs you of the user’s current user interface theme.

Use the `describeQuickActions()`, `describeTabs()`, and `describeTheme()` calls and their return types to get information on theme icons and colors.

Style sheets are available to mimic the look and feel of the older user interfaces. For more information, see [Styling Visualforce Pages](#) in the *Visualforce Developer’s Guide*. But if you’re planning to switch to Lightning Experience, consider the Lightning Component framework, our new UI framework. See the “[Lightning Components](#)” module in the [Develop for Lightning Experience](#) Trailhead trail to learn more.

EDITIONS

Available in: Salesforce Classic and earlier

Examples Using the Partner WSDL

This section includes examples in Java and C# for making API calls using the partner WSDL. Before running these samples, perform the following steps in the quick start tutorial to get the partner WSDL file and generate the proxy client code for your development environment.

- [Step 2: Generate or Obtain the Web Service WSDL](#)
- [Step 3: Import the WSDL File Into Your Development Platform](#)

After you generate the proxy client code and set up your development environment, you can start writing your client application. First, your application needs to log into the Salesforce service using the partner authentication endpoint. After a successful login, you can execute the sample methods.

For your convenience, template classes are provided, one in Java and one in C#, that make a login call. You can use them to execute the sample methods provided later in this section.

Sample template class for Java: This sample prompts the user to enter the username, password, and authentication endpoint. Next, it logs the user in. For the authentication endpoint URL, pass in the endpoint found in the partner WSDL file.

```
import com.sforce.soap.partner.PartnerConnection;
import com.sforce.soap.partner.subject.*;
import com.sforce.soap.partner.*;
import com.sforce.ws.ConnectorConfig;
import com.sforce.ws.ConnectionException;
import com.sforce.soap.partner.Error;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.util.*;

public class PartnerSamples {
    PartnerConnection partnerConnection = null;
    private static BufferedReader reader =
        new BufferedReader(new InputStreamReader(System.in));

    public static void main(String[] args) {
        PartnerSamples samples = new PartnerSamples();
        if (samples.login()) {
            // Add calls to the methods in this class.
            // For example:
            // samples.querySample();
        }
    }

    private String getUserInput(String prompt) {
        String result = "";
        try {
            System.out.print(prompt);
            result = reader.readLine();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
        return result;
    }

    private boolean login() {
        boolean success = false;
        String username = getUserInput("Enter username: ");
        String password = getUserInput("Enter password: ");
        String authEndPoint = getUserInput("Enter auth end point: ");

        try {
            ConnectorConfig config = new ConnectorConfig();
            config.setUsername(username);
            config.setPassword(password);

            config.setAuthEndpoint(authEndPoint);
```

```

        config.setTraceFile("traceLogs.txt");
        config.setTraceMessage(true);
        config.setPrettyPrintXml(true);

        partnerConnection = new PartnerConnection(config);

        success = true;
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    } catch (FileNotFoundException fnfe) {
        fnfe.printStackTrace();
    }

    return success;
}

//
// Add your methods here.
//
}

```

Sample template class for C#: This sample prompts the user to enter the username and password. Next, it logs the user in. The project name for this sample is assumed to be *TemplatePartner* and the Web reference name *sforce*. If these values are different for your project, make sure to change the using directive to appropriate values for your project: `using your_project_name.web_reference_name;`

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.Services.Protocols;
using System.Collections;
using TemplatePartner.sforce;

namespace TemplatePartner
{
    class PartnerSamples
    {
        private SforceService binding;

        static void Main(string[] args)
        {
            PartnerSamples samples = new PartnerSamples();
            if (samples.login())
            {
                // Add calls to the methods in this class.
                // For example:
                // samples.querySample();
            }

            private bool login()
            {
                Console.Write("Enter username: ");
            }
        }
    }
}

```

```
string username = Console.ReadLine();
Console.Write("Enter password: ");
string password = Console.ReadLine();

// Create a service object
binding = new SforceService();

// Timeout after a minute
binding.Timeout = 60000;

// Try logging in
LoginResult lr;
try
{
    Console.WriteLine("\nLogging in...\n");
    lr = binding.login(username, password);
}

// ApiFault is a proxy stub generated from the WSDL contract when
// the web service was imported
catch (SoapException e)
{
    // Write the fault code to the console
    Console.WriteLine(e.Code);

    // Write the fault message to the console
    Console.WriteLine("An unexpected error has occurred: " + e.Message);

    // Write the stack trace to the console
    Console.WriteLine(e.StackTrace);

    // Return False to indicate that the login was not successful
    return false;
}

// Check if the password has expired
if (lr.passwordExpired)
{
    Console.WriteLine("An error has occurred. Your password has expired.");
    return false;
}

// Set the returned service endpoint URL
binding.Url = lr.serverUrl;

// Set the SOAP header with the session ID returned by
// the login result. This will be included in all
// API calls.
binding.SessionHeaderValue = new SessionHeader();
binding.SessionHeaderValue.sessionId = lr.sessionId;

// Return true to indicate that we are logged in, pointed
// at the right URL and have our security token in place.
```

```

        return true;
    }

    //
    // Add your methods here.
    //
}

```

This partner WSDL samples are:

- [Sample query and queryMore Calls](#)
- [Sample search Call](#)
- [Sample create Call](#)
- [Sample update Call](#)

Sample query and queryMore Calls

The following Java and C# examples show usage of the `query()` and `queryMore()` calls for the partner WSDL. Each example sets the batch size of the query to 250 items returned. It then performs a query call to get the first name and last name of all contacts and iterates through the contact records returned. For each contact, it writes the contact's first name and last name to the output, or only the last name if the first name is null. Finally, if there are more items to be returned by the query, as indicated by a `QueryResult.done` property value of `false`, it calls `queryMore()` to get the next batch of items, and repeats the process until no more records are returned.

To execute the sample method, you can use the corresponding Java or C# template class provided in [Examples Using the Partner WSDL](#).

Java Example

```

public void querySample() {
    try {
        // Set query batch size
        partnerConnection.setQueryOptions(250);

        // SQL query to use
        String soqlQuery = "SELECT FirstName, LastName FROM Contact";
        // Make the query call and get the query results
        QueryResult qr = partnerConnection.query(soqlQuery);

        boolean done = false;
        int loopCount = 0;
        // Loop through the batches of returned results
        while (!done) {
            System.out.println("Records in results set " + loopCount++
                + " - ");
            SObject[] records = qr.getRecords();
            // Process the query results
            for (int i = 0; i < records.length; i++) {
                SObject contact = records[i];
                Object firstName = contact.getField("FirstName");
                Object lastName = contact.getField("LastName");
                if (firstName == null) {
                    System.out.println("Contact " + (i + 1) +

```



```

                ": " + lastName
            );
        } else {
            System.out.println("Contact " + (i + 1) + ": " +
                firstName + " " + lastName);
        }
    }
    if (qr.isDone()) {
        done = true;
    } else {
        qr = partnerConnection.queryMore(qr.getQueryLocator());
    }
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
System.out.println("\nQuery execution completed.");
}

```

C# Example

```

public void querySample()
{
    try
    {
        QueryResult qr = null;
        binding.QueryOptionsValue = new sforce.QueryOptions();
        binding.QueryOptionsValue.batchSize = 250;
        binding.QueryOptionsValue.batchSizeSpecified = true;

        qr = binding.query("SELECT FirstName, LastName FROM Contact");

        bool done = false;
        int loopCount = 0;
        while (!done)
        {
            Console.WriteLine("\nRecords in results set " +
                Convert.ToString(loopCount++)
                + " - ");
            // Process the query results
            for (int i = 0; i < qr.records.Length; i++)
            {
                sforce.sObject con = qr.records[i];
                string fName = con.Any[0].InnerText;
                string lName = con.Any[1].InnerText;
                if (fName == null)
                    Console.WriteLine("Contact " + (i + 1) + ": " + lName);
                else
                    Console.WriteLine("Contact " + (i + 1) + ": " + fName
                        + " " + lName);
            }

            if (qr.done)

```

```

        done = true;
    else
        qr = binding.queryMore(qr.queryLocator);
    }
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " + e.Message +
        " Stack trace: " + e.StackTrace);
}
Console.WriteLine("\nQuery execution completed.");
}

```

Sample search Call

The following Java and C# examples show how to use the `search()` call for the partner WSDL. Each example accepts a phone number string value that is used in the SOQL query. The search call looks for phone fields that match the passed in phone value in all contacts, leads, and accounts. Next, the example iterates through the returned search results that contain the matching records, adds them to arrays, and writes their field values to the console. The record fields returned correspond to the fields specified in the SOQL query for each record type.

To execute the sample method, you can use the corresponding Java or C# template class provided in [Examples Using the Partner WSDL](#).

Java Example

```

public void searchSample(String phoneNumber) {
    try {
        // Example of phoneNumber format: 4155551212
        String soslQuery =
            "FIND {" + phoneNumber + "} IN Phone FIELDS " +
            "RETURNING " +
            "Contact(Id, Phone, FirstName, LastName), " +
            "Lead(Id, Phone, FirstName, LastName)," +
            "Account(Id, Phone, Name)";
        // Perform SOSL query
        SearchResult sResult = partnerConnection.search(soslQuery);
        // Get the records returned by the search result
        SearchRecord[] records = sResult.getSearchRecords();
        // Create lists of objects to hold search result records
        List<SObject> contacts = new ArrayList<SObject>();
        List<SObject> leads = new ArrayList<SObject>();
        List<SObject> accounts = new ArrayList<SObject>();

        // Iterate through the search result records
        // and store the records in their corresponding lists
        // based on record type.
        if (records != null && records.length > 0) {
            for (int i = 0; i < records.length; i++){
                SObject record = records[i].getRecord();
                if (record.getType().toLowerCase().equals("contact")) {
                    contacts.add(record);
                } else if (record.getType().toLowerCase().equals("lead")){

```

```

        leads.add(record);
    } else if (record.getType().toLowerCase().equals("account")) {
        accounts.add(record);
    }
}
// Display the contacts that the search returned
if (contacts.size() > 0) {
    System.out.println("Found " + contacts.size() +
        " contact(s):");
    for (SObject contact : contacts) {
        System.out.println(contact.getId() + " - " +
            contact.getField("FirstName") + " " +
            contact.getField("LastName") + " - " +
            contact.getField("Phone")
        );
    }
}
// Display the leads that the search returned
if (leads.size() > 0) {
    System.out.println("Found " + leads.size() +
        " lead(s):");
    for (SObject lead : leads) {
        System.out.println(lead.getId() + " - " +
            lead.getField("FirstName") + " " +
            lead.getField("LastName") + " - " +
            lead.getField("Phone")
        );
    }
}
// Display the accounts that the search returned
if (accounts.size() > 0) {
    System.out.println("Found " +
        accounts.size() + " account(s):");
    for (SObject account : accounts) {
        System.out.println(account.getId() + " - " +
            account.getField("Name") + " - " +
            account.getField("Phone")
        );
    }
}
} else {
    // The search returned no records
    System.out.println("No records were found for the search.");
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

C# Example

```

public void searchSample(String phoneNumber)
{

```

```

try
{
    // Example of phoneNumber format: 4155551212
    String soslQuery =
        "FIND {" + phoneNumber + "} IN Phone FIELDS " +
        "RETURNING " +
        "Contact(Id, Phone, FirstName, LastName), " +
        "Lead(Id, Phone, FirstName, LastName)," +
        "Account(Id, Phone, Name)";
    // Perform SOSL query
    SearchResult sResult = binding.search(soslQuery);
    // Get the records returned by the search result
    SearchRecord[] records = sResult.searchRecords;
    // Create lists of objects to hold search result records
    ArrayList contacts = new System.Collections.ArrayList();
    ArrayList leads = new System.Collections.ArrayList();
    ArrayList accounts = new System.Collections.ArrayList();

    // Iterate through the search result records
    // and store the records in their corresponding lists
    // based on record type.
    if ((records != null) && (records.Length > 0))
    {
        for (int i = 0; i < records.Length; i++)
        {
            sObject record = records[i].record;

            if (record.type.ToLower().Equals("contact"))
            {
                contacts.Add(record);
            }
            else if (record.type.ToLower().Equals("lead"))
            {
                leads.Add(record);
            }
            else if (record.type.ToLower().Equals("account"))
            {
                accounts.Add(record);
            }
        }
    }
    // Display the contacts that the search returned
    if (contacts.Count > 0)
    {
        Console.WriteLine("Found " + contacts.Count + " contact(s):");
        for (int i = 0; i < contacts.Count; i++)
        {
            sObject c = (sObject)contacts[i];
            Console.WriteLine(c.Any[0].InnerText + " - " +
                c.Any[2].InnerText + " " +
                c.Any[3].InnerText + " - " + c.Any[1].InnerText);
        }
    }
    // Display the leads that the search returned
    if (leads.Count > 0)

```

```

    {
        Console.WriteLine("Found " + leads.Count + " lead(s):");
        for (int i = 0; i < leads.Count; i++)
        {
            sObject l = (sObject)leads[i];
            Console.WriteLine(l.Any[0].InnerText + " - " +
                l.Any[2].InnerText + " " +
                l.Any[3].InnerText + " - " + l.Any[1].InnerText);
        }
    }
    // Display the accounts that the search returned
    if (accounts.Count > 0)
    {
        Console.WriteLine("Found " + accounts.Count + " account(s):");
        for (int i = 0; i < accounts.Count; i++)
        {
            sObject a = (sObject)accounts[i];
            Console.WriteLine(a.Any[0].InnerText + " - " +
                a.Any[2].InnerText + " - " +
                a.Any[1].InnerText);
        }
    }
}
else
{
    // The search returned no records
    Console.WriteLine("No records were found for the search.");
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " + e.Message +
        " Stack trace: " + e.StackTrace);
}
}

```

Sample create Call

The following Java and C# examples show how to use the `create()` call for the partner WSDL. Each example creates a contact record with several fields. It iterates through the results of the create call and checks whether the operation was successful or not. If the create operation was successful, it writes the ID of the contact created to the console. Otherwise, it iterates through the errors and writes details of each error to the console. In this case, the output of the example is the ID of the new contact.

To execute the sample method, you can use the corresponding Java or C# template class provided in [Examples Using the Partner WSDL](#).

Java Example

```

public String createSample() {
    String result = null;
    try {
        // Create a new sObject of type Contact
        // and fill out its fields.
        SObject contact = new SObject();
    }
}

```

```

contact.setType("Contact");
contact.setField("FirstName", "Otto");
contact.setField("LastName", "Jespersen");
contact.setField("Salutation", "Professor");
contact.setField("Phone", "(999) 555-1234");
contact.setField("Title", "Philologist");

// Add this sObject to an array
SObject[] contacts = new SObject[1];
contacts[0] = contact;
// Make a create call and pass it the array of sObjects
SaveResult[] results = partnerConnection.create(contacts);

// Iterate through the results list
// and write the ID of the new sObject
// or the errors if the object creation failed.
// In this case, we only have one result
// since we created one contact.
for (int j = 0; j < results.length; j++) {
    if (results[j].isSuccess()) {
        result = results[j].getId();
        System.out.println(
            "\nA contact was created with an ID of: " + result
        );
    } else {
        // There were errors during the create call,
        // go through the errors array and write
        // them to the console
        for (int i = 0; i < results[j].getErrors().length; i++) {
            Error err = results[j].getErrors()[i];
            System.out.println("Errors were found on item " + j);
            System.out.println("Error code: " +
                err.getStatusCode().toString());
            System.out.println("Error message: " + err.getMessage());
        }
    }
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
return result;
}

```

C# Example

```

public void createSample()
{
    try
    {
        // Create a new sObject of type Contact
        // and fill out its fields.
        sObject contact = new sforce.sObject();
        System.Xml.XmlElement[] contactFields = new System.Xml.XmlElement[6];
    }
}

```

```

// Create the contact's fields
System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
contactFields[0] = doc.CreateElement("FirstName");
contactFields[0].InnerText = "Otto";
contactFields[1] = doc.CreateElement("LastName");
contactFields[1].InnerText = "Jespersen";
contactFields[2] = doc.CreateElement("Salutation");
contactFields[2].InnerText = "Professor";
contactFields[3] = doc.CreateElement("Phone");
contactFields[3].InnerText = "(999) 555-1234";
contactFields[4] = doc.CreateElement("Title");
contactFields[4].InnerText = "Philologist";

contact.type = "Contact";
contact.Any = contactFields;

// Add this sObject to an array
sObject[] contactList = new sObject[1];
contactList[0] = contact;

// Make a create call and pass it the array of sObjects
SaveResult[] results = binding.create(contactList);
// Iterate through the results list
// and write the ID of the new sObject
// or the errors if the object creation failed.
// In this case, we only have one result
// since we created one contact.
for (int j = 0; j < results.Length; j++)
{
    if (results[j].success)
    {
        Console.WriteLine("\nA contact was created with an ID of: "
            + results[j].id);
    }
    else
    {
        // There were errors during the create call,
        // go through the errors array and write
        // them to the console
        for (int i = 0; i < results[j].errors.Length; i++)
        {
            Error err = results[j].errors[i];
            Console.WriteLine("Errors were found on item " + j.ToString());
            Console.WriteLine("Error code is: " + err.statusCode.ToString());
            Console.WriteLine("Error message: " + err.message);
        }
    }
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " + e.Message +
        " Stack trace: " + e.StackTrace);
}

```

```

    }
}

```

Sample update Call

The following Java and C# examples show how to use the `update()` call for the Partner WSDL. Each example takes the ID of the contact to update as an argument. It creates two `sObject` records of type `Contact`—one to hold the valid passed in ID and the other has an invalid ID. Next, it sets a new phone number for the valid contact and `null` for the last name of the invalid contact. It then makes the update call and iterates through the results. For a successful update operation, it writes the ID of the contact that got updated. For a failed update operation, it writes the details of all returned errors to the console. In this case, the output is the ID of the contact that was successfully updated and an error for the invalid contact update.

To execute the sample method, you can use the corresponding Java or C# template class provided in [Examples Using the Partner WSDL](#).

Java Example

```

public void updateSample(String id) {
    try {
        // Create an sObject of type contact
        SObject updateContact = new SObject();
        updateContact.setType("Contact");

        // Set the ID of the contact to update
        updateContact.setId(id);
        // Set the Phone field with a new value
        updateContact.setField("Phone", "(415) 555-1212");

        // Create another contact that will cause an error
        // because it has an invalid ID.
        SObject errorContact = new SObject();
        errorContact.setType("Contact");
        // Set an invalid ID on purpose
        errorContact.setId("SLFKJLFKJ");
        // Set the value of LastName to null
        errorContact.setFieldsToNull(new String[] {"LastName"});

        // Make the update call by passing an array containing
        // the two objects.
        SaveResult[] saveResults = partnerConnection.update(
            new SObject[] {updateContact, errorContact}
        );
        // Iterate through the results and write the ID of
        // the updated contacts to the console, in this case one contact.
        // If the result is not successful, write the errors
        // to the console. In this case, one item failed to update.
        for (int j = 0; j < saveResults.length; j++) {
            System.out.println("\nItem: " + j);
            if (saveResults[j].isSuccess()) {
                System.out.println("Contact with an ID of " +
                    saveResults[j].getId() + " was updated.");
            }
            else {

```



```

        // There were errors during the update call,
        // go through the errors array and write
        // them to the console.
        for (int i = 0; i < saveResults[j].getErrors().length; i++) {
            Error err = saveResults[j].getErrors()[i];
            System.out.println("Errors were found on item " + j);
            System.out.println("Error code: " +
                err.getStatusCode().toString());
            System.out.println("Error message: " + err.getMessage());
        }
    }
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

For more information about `setFieldsToNull` (or its equivalent in client tools other than WSC), see [fieldsToNull](#) and [Resetting Values to null](#).

C# Example

```

public void updateSample(String id) {
    try
    {
        // Create an sObject of type contact
        sObject updateContact = new sObject();
        updateContact.type = "Contact";

        // Set the ID of the contact to update
        updateContact.Id = id;
        // Set the Phone field to a new value.
        // The Phone field needs to be created as an XML element.
        System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
        System.Xml.XmlElement phoneField = doc.CreateElement("Phone");
        phoneField.InnerText = "(415) 555-1212";

        // Add the Phone field to the contact
        updateContact.Any = new System.Xml.XmlElement[] {phoneField};

        // Create another contact that will cause an error
        // because it has an invalid ID.
        sObject errorContact = new sObject();
        errorContact.type = "Contact";
        // Set an invalid ID on purpose
        errorContact.Id = "SLFKJLFKJ";
        // Set the value of LastName to null
        errorContact.fieldsToNull = new String[] { "LastName" };

        // Make the update call by passing an array containing
        // the two objects.
        SaveResult[] saveResults = binding.update(
            new sObject[] {updateContact, errorContact});
        // Iterate through the results and write the ID of
    }
}

```

```
// the updated contacts to the console, in this case one contact.
// If the result is not successful, write the errors
// to the console. In this case, one item failed to update.
for (int j = 0; j < saveResults.Length; j++) {
    Console.WriteLine("\nItem: " + j);
    if (saveResults[j].success)
    {
        Console.WriteLine("Contact with an ID of " +
            saveResults[j].id + " was updated.");
    }
    else
    {
        // There were errors during the update call,
        // go through the errors array and write
        // them to the console.
        for (int i = 0; i < saveResults[j].errors.Length; i++) {
            Error err = saveResults[j].errors[i];
            Console.WriteLine("Errors were found on item " + j.ToString());
            Console.WriteLine("Error code: " +
                err.statusCode.ToString());
            Console.WriteLine("Error message: " + err.message);
        }
    }
}
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " + e.Message +
        " Stack trace: " + e.StackTrace);
}
}
```

CHAPTER 8 Apex-Related Calls

In this chapter ...

- [compileAndTest\(\)](#)
- [compileClasses\(\)](#)
- [compileTriggers\(\)](#)
- [executeAnonymous\(\)](#)
- [runTests\(\)](#)

The following table lists supported calls in the API in alphabetical order, and provides a brief description for each. Click a call name to see syntax, usage, and more information for that call.



Note: For a list of core calls, see [Core Calls](#), for a list of describe calls, see [Describe Calls](#), and for a list of utility calls, see [Utility Calls](#).

Call	Description
compileAndTest()	Compile and test your Apex in a single call.
compileClasses()	Compile your Apex in Developer Edition or sandbox organizations.
compileTriggers()	Compile your Apex triggers in Developer Edition or sandbox organizations.
executeAnonymous()	Execute a block of Apex.
runTests()	Run your Apex unit tests.

compileAndTest()

Compile and test your Apex in a single call.

Syntax

```
CompileAndTestResult[] = compileAndTest(CompileAndTestRequest request);
```

Usage

Use this call to both compile and test the Apex you specify with a single call. Production organizations (not a Developer Edition or Sandbox Edition) must use this call instead of [compileClasses\(\)](#) or [compileTriggers\(\)](#).

This call supports the [DebuggingHeader](#) on page 342 and the [SessionHeader](#) on page 356.

All specified tests must pass, otherwise data is not saved to the database. If this call is invoked in a production organization, the [RunTestsRequest](#) property of the [CompileAndTestRequest](#) is ignored, and all unit tests defined in the organization are run and must pass.

Sample Code—Java

Note that the following example sets `checkOnly` to `true` so that this class is compiled and tested, but the classes are not saved to the database.

```
{
    CompileAndTestRequest request;
    CompileAndTestResult result = null;

    String triggerBody = "trigger t1 on Account (before insert){ " +
        "    for(Account a:Trigger.new){ " +
        "        a.description = 't1_UPDATE'};" +
        "    }";

    String testClassBody = "@isTest private class TestT1{" +
        "    // Test for the trigger" +
        "    public static testmethod void test1(){ " +
        "        Account a = new Account(name='TEST');" +
        "        insert(a);" +
        "        a = [select id,description from Account where id=:a.id];" +
        "        System.assert(a.description.contains('t1_UPDATE'));" +
        "    }" +
        "    // Test for the class" +
        "    public static testmethod void test2(){ " +
        "        String s = Cl.method1();" +
        "        System.assert(s=='HELLO'));" +
        "    }" +
        "    }";

    String classBody = "public class Cl{" +
        "    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 <code>true</code> , the Apex classes and triggers submitted are not saved to your organization, whether or not the code successfully compiles and unit tests pass.
classes	string	Content of the class or classes to be compiled.
deleteClasses	string	Name of the class or classes to be deleted.
deleteTriggers	string	Name of the trigger or triggers to be deleted.
runTestsRequest	RunTestsRequest	Specifies information about the Apex to be tested. If this request is sent in a production organization, this property is ignored and all unit tests are run for your entire organization.
triggers	string	Content of the trigger or triggers to be compiled.

Note the following about this object:

- This object contains the [RunTestsRequest](#) property. If the request is run in a production organization, the property is ignored and all tests are run.
- If any errors occur during compile, delete, testing, or if the goal of 75% code coverage is missed, no classes or triggers are saved to your organization. This is the same requirement as Salesforce AppExchange package testing.
- All triggers must have code coverage. If a trigger has no code coverage, no classes or triggers are saved to your organization.

CompileAndTestResult

The [compileAndTest \(\)](#) call returns information about the compile and unit test run of the specified Apex, including whether it succeeded or failed.

A [CompileAndTestResult](#) object has the following properties:

Name	Type	Description
<code>classes</code>	CompileClassResult	Information about the success or failure of the compileAndTest () call if classes were being compiled.
<code>deleteClasses</code>	DeleteApexResult	Information about the success or failure of the compileAndTest () call if classes were being deleted.
<code>deleteTriggers</code>	DeleteApexResult	Information about the success or failure of the compileAndTest () call if triggers were being deleted.
<code>runTestsResult</code>	RunTestsResult	Information about the success or failure of the Apex unit tests, if any were specified.
<code>success</code>	boolean	If <code>true</code> , all of the classes, triggers, and unit tests specified ran successfully. If any class, trigger, or unit test failed, the value is <code>false</code> , and details are reported in the corresponding result object: <ul style="list-style-type: none"> • CompileClassResult • CompileTriggerResult • DeleteApexResult • RunTestsResult
<code>triggers</code>	CompileTriggerResult	Information about the success or failure of the compileAndTest () call if triggers were being compiled.

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
<code>bodyCrc</code>	int	The CRC (cyclic redundancy check) of the class or trigger file.

Name	Type	Description
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 <code>true</code> , the class or classes compiled successfully. If <code>false</code> , problems are specified in other properties of this object.

CompileTriggerResult

This object is returned as part of a `compileAndTest()` or `compileTriggers()` call. It contains information about whether or not the compile and run of the specified Apex was successful.

A `CompileTriggerResult` object has the following properties:

Name	Type	Description
bodyCrc	int	The CRC (cyclic redundancy check) of the trigger file.
column	int	The column where an error occurred, if one did.
id	ID	An ID is created for each compiled trigger. The ID is unique within an organization.
line	int	The line number where an error occurred, if one did.
name	string	The name of the trigger.
problem	string	The description of the problem if an error occurred.
success	boolean	If <code>true</code> , all the specified triggers compiled and ran successfully. If the compilation or execution of any trigger fails, the value is <code>false</code> .

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 <code>true</code> , all the specified classes or triggers were deleted successfully. If any class or trigger is not deleted, the value is <code>false</code> .

compileClasses()

Compile your Apex in Developer Edition or sandbox organizations.

Syntax

```
CompileClassResult[] = compileClasses(string[] classList);
```

Usage

Use this call to compile Apex classes in Developer Edition or sandbox organizations. Production organizations must use [compileAndTest\(\)](#).

This call supports the [DebuggingHeader](#) on page 342 and the [SessionHeader](#) on page 356.

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.

Response

[CompileClassResult](#)

compileTriggers ()

Compile your Apex triggers in Developer Edition or sandbox organizations.

Syntax

```
CompileTriggerResult[] = compileTriggers(string[] triggerList);
```

Usage

Use this call to compile the specified Apex triggers in your Developer Edition or sandbox organization. Production organizations must use [compileAndTest \(\)](#).

This call supports the [DebuggingHeader](#) on page 342 and the [SessionHeader](#) on page 356.

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.

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](#) on page 342 and [SessionHeader](#) on page 356.

If a component in a package with restricted API access issues this call, the request is blocked.

Apex classes and triggers saved (compiled) using API version 15.0 and higher produce a runtime error if you assign a String value that is too long for the field.

Arguments

Name	Type	Description
apexcode	string	A block of Apex.

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 <code>compiled</code> is False, this field contains the column number of the point where the compile failed.
compileProblem	string	If <code>compiled</code> 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 <code>column</code> , <code>line</code> , and <code>compileProblem</code> fields are not null.
exceptionMessage	string	If <code>success</code> is False, this field contains the exception message for the failure.
exceptionStackTrace	string	If <code>success</code> is False, this field contains the stack trace for the failure.
line	int	If <code>compiled</code> 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 <code>exceptionMessage</code> and <code>exceptionStackTrace</code> values are not null.

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, and send no emails. Such methods are flagged with the `@isTest` annotation in the method definition. Unit 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](#) on page 342 and the [SessionHeader](#) on page 356.

Sample Code—Java

```
public void runTestsSample() {
    String sessionId = "sessionId goes here";
    String url = "url goes here";
    // Set the Apex stub with session ID received from logging in with the partner API
    _SessionHeader sh = new _SessionHeader();
    apexBinding.setHeader(
        new ApexServiceLocator().getServiceName().getNamespaceURI(),
        "SessionHeader", sh);
    // Set the URL received from logging in with the partner API to the Apex stub
    apexBinding._setProperty(ApexBindingStub.ENDPOINT_ADDRESS_PROPERTY, url);

    // Set the debugging header
    _DebuggingHeader dh = new _DebuggingHeader();
    dh.setDebugLevel(LogType.Profiling);
    apexBinding.setHeader(
        new ApexServiceLocator().getServiceName().getNamespaceURI(),
        "DebuggingHeader", dh);

    long start = System.currentTimeMillis();
    RunTestsRequest rtr = new RunTestsRequest();
    rtr.setAllTests(true);
    RunTestsResult res = null;
    try {
        res = apexBinding.runTests(rtr);
    } catch (RemoteException e) {
        System.out.println("An unexpected error occurred: " + e.getMessage());
    }

    System.out.println("Number of tests: " + res.getNumTestsRun());
    System.out.println("Number of failures: " + res.getNumFailures());
    if (res.getNumFailures() > 0) {
        for (RunTestFailure rtf : res.getFailures()) {
            System.out.println("Failure: " + (rtf.getNamespace() ==
                null ? "" : rtf.getNamespace() + ".")
                + rtf.getName() + "." + rtf.getMethodName() + ": "
                + rtf.getMessage() + "\n" + rtf.getStackTrace());
        }
    }
}
```

```

}
if (res.getCodeCoverage() != null) {
    for (CodeCoverageResult ccr : res.getCodeCoverage()) {
        System.out.println("Code coverage for " + ccr.getType() +
            (ccr.getNamespace() == null ? "" : ccr.getNamespace() + ".")
            + ccr.getName() + ": "
            + ccr.getNumLocationsNotCovered()
            + " locations not covered out of "
            + ccr.getNumLocations());

        if (ccr.getNumLocationsNotCovered() > 0) {
            for (CodeLocation cl : ccr.getLocationsNotCovered())
                System.out.println("\tLine " + cl.getLine());
        }
    }
}
System.out.println("Finished in " +
    (System.currentTimeMillis() - start) + "ms");
}

```

Arguments

Name	Type	Description
request	RunTestsRequest	A request that includes the Apex unit tests and the values for any fields that need to be set for this request.

Response

[RunTestsResult](#)

RunTestsRequest

Specifies information about the Apex code to be tested. `RunTestsRequest` is part of `CompileAndTestRequest`, which is the request passed to the `compileAndTest()` call. This object is also passed to the Tooling SOAP API call `runTests()`. You can specify the same or different classes to be tested and compiled. Since triggers cannot be tested directly, they are not included in this object. Instead, you must specify a class that calls the trigger.

If the request is sent to a production organization, this request is ignored and all unit tests defined for your organization are run.

The `RunTestsRequest` object has the following properties:

Name	Type	Description
<code>allTests</code>	boolean	If <code>allTests</code> is true, all unit tests defined for your organization are run.
<code>classes</code>	string[]	An array of one or more objects.
<code>namespace</code>	string	If specified, the namespace that contains the unit tests to be run. Do not use this property if you specify <code>allTests</code> as true. Also, if you execute <code>compileAndTest()</code>

Name	Type	Description
		in a production organization, this property is ignored, and all unit tests defined for the organization are run.
<code>maxFailedTests</code>	<code>int</code>	A mandatory parameter for the Tooling SOAP API call <code>runTests()</code> . To allow all tests in a run to execute, set <code>maxFailedTests</code> to <code>-1</code> . To stop the test run from executing new tests after a given number of tests fail, set <code>maxFailedTests</code> to an integer value from <code>0</code> to <code>1,000,000</code> . This integer value sets the maximum allowable test failures. A value of <code>0</code> causes the test run to stop if any failure occurs. A value of <code>1</code> causes the test run to stop on the second failure, and so on.
<code>packages</code>	<code>string[]</code>	Do not use after version 10.0. For earlier, unsupported releases, the content of the package to be tested.
<code>skipCodeCoverage</code>	<code>boolean</code>	Indicates whether to opt out of collecting code coverage information during Apex test runs. Available in API version 43.0 and later.
<code>tests</code>	TestsNode[]	A mandatory parameter for the Tooling SOAP API call <code>runTests()</code> . Specifies individual test methods in an Apex test class. To specify classes or suites instead of a <code>TestsNode[]</code> , set <code>tests</code> to <code>null</code> . Although this property accepts an array, the array can contain only one entry.

TestsNode

Specifies individual test methods in an Apex test class.

Name	Type	Description
<code>classId</code>	<code>string</code>	<p>Description</p> <p>The ID of the Apex class that contains the test methods you want to run.</p> <p><code>classId</code> or <code>className</code> is required.</p> <p>Supported Methods</p> <ul style="list-style-type: none"> <code>getClassId()</code> <code>setClassId(new String "<your class ID>")</code>
<code>className</code>	<code>string</code>	<p>Description</p> <p>The name of the Apex class that contains the test methods you want to run.</p> <p>To run tests from a managed package, include the package's namespace using dot notation.</p> <p><code>classId</code> or <code>className</code> is required.</p> <p>Supported Methods</p> <ul style="list-style-type: none"> <code>getClassName()</code> <code>setClassName(new String "YourClassName")</code>

Name	Type	Description
testMethods	string[]	<p>Description</p> <p>The test methods you want to run.</p> <p>Required.</p> <p>Supported Methods</p> <ul style="list-style-type: none"> • <code>getTestMethods()</code> • <code>setTestMethods(new String[] { "testMethod1", "testMethod2" })</code>

RunTestsResult

Contains information about the execution of unit tests, including whether unit tests were completed successfully, code coverage results, and failures.

A RunTestsResult object has the following properties:

Name	Type	Description
apexLogId	string	The ID of an ApexLog object that is created at the end of a test run. The ApexLog object is created if there is an active trace flag on the user running an Apex test, or on a class or trigger being executed. This field is available in API version 35.0 and later.
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.
flowCoverage	FlowCoverageResult on page 103[]	An array of results from test runs that executed flows. This field is available in API version 44.0 and later.
flowCoverageWarnings	FlowCoverageWarning on page 103[]	An array of warnings generated by test runs that executed flows. This field is available in API version 44.0 and later.
numFailures	int	The number of failures for the unit tests.
numTestsRun	int	The number of unit tests that were run.

Name	Type	Description
successes	RunTestSuccess []	An array of one or more RunTestSuccess objects that contain information about successes, if there are any.
totalTime	double	The total cumulative time spent running tests, in milliseconds. This can be helpful for performance monitoring.

CodeCoverageResult

The [RunTestsResult](#) object contains this object. It contains information about whether or not the compile of the specified Apex and run of the unit tests was successful.

A [CodeCoverageResult](#) object has the following properties:

Name	Type	Description
dmlInfo	CodeLocation []	For each class or trigger tested, for each portion of code tested, this property contains the DML statement locations, the number of times the code was executed, and the total cumulative time spent in these calls. This can be helpful for performance monitoring.
id	ID	The ID of the CodeLocation . The ID is unique within an organization.
locationsNotCovered	CodeLocation []	For each class or trigger tested, if any code is not covered, the line and column of the code not tested, and the number of times the code was executed.
methodInfo	CodeLocation []	For each class or trigger tested, the method invocation locations, the number of times the code was executed, and the total cumulative time spent in these calls. This can be helpful for performance monitoring.
name	string	The name of the class or trigger covered.
namespace	string	The namespace that contained the unit tests, if one is specified.
numLocations	int	The total number of code locations.
soqlInfo	CodeLocation []	For each class or trigger tested, the location of SOQL statements in the code, the number of times this code was executed, and the total cumulative time spent in these calls. This can be helpful for performance monitoring.
soslInfo	CodeLocation []	For each class tested, the location of SOSL statements in the code, the number of times this code was executed, and the total cumulative time spent in these calls. This can be helpful for performance monitoring.
type	string	Do not use. In early, unsupported releases, used to specify class or package.

CodeCoverageWarning

The [RunTestsResult](#) object contains this object. It contains information about the Apex class which generated warnings.

This object has the following properties:

Name	Type	Description
id	ID	The ID of the class which generated warnings.
message	string	The message of the warning generated.
name	string	The name of the class that generated a warning. If the warning applies to the overall code coverage, this value is null.
namespace	string	The namespace that contains the class, if one was specified.

RunTestFailure

The [RunTestsResult](#) object returns information about failures during the unit test run.

This object has the following properties:

Name	Type	Description
id	ID	The ID of the class which generated failures.
message	string	The failure message.
methodName	string	The name of the method that failed.
name	string	The name of the class that failed.
namespace	string	The namespace that contained the class, if one was specified.
seeAllData	boolean	Indicates whether the test method has access to organization data (true) or not (false). This field is available in API version 33.0 and later.
stackTrace	string	The stack trace for the failure.
time	double	The time spent running tests for this failed operation, in milliseconds. This can be helpful for performance monitoring.
type	string	Do not use. In early, unsupported releases, used to specify class or package.

FlowCoverageResult

This object contains information about the flow version and the number of elements executed by the test run. This object is available in API version 44.0 and later.

Name	Type	Description
elementsNotCovered	string	List of elements in the flow version that weren't executed by the test run.
flowId	string	The ID of the flow version. The ID is unique within an org.
flowName	string	The name of the flow that was executed by the test run.
flowNamespace	string	The namespace that contains the flow, if one is specified.
numElements	int	The total number of elements in the flow version.
numElementsNotCovered	int	The number of elements in the flow version that weren't executed by the test run
processType	FlowProcessType (enumeration of type string)	The process type of the flow version.

FlowCoverageWarning

This object contains information about the flow version that generated warnings. This object is available in API version 44.0 and later.

Name	Type	Description
flowId	string	The ID of the flow version that generated the warning.
flowName	string	The name of the flow that generated the warning. If the warning applies to the overall test coverage of flows within your org, this value is null.
flowNamespace	string	The namespace that contains the flow, if one was specified.
message	string	The message of the warning that was generated.

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.

Name	Type	Description
namespace	string	The namespace that contained the class, if one was specified.
seeAllData	boolean	Indicates whether the test method has access to organization data (<code>true</code>) or not (<code>false</code>). This field is available in API version 33.0 and later.
time	double	The time spent running tests for this operation. This can be helpful for performance monitoring.

CodeLocation


The [RunTestsResult](#) object contains this object in a number of fields.

This object has the following properties:

Name	Type	Description
column	int	The column location of the Apex tested.
line	int	The line location of the Apex tested.
numExecutions	int	The number of times the Apex was executed in the test run.
time	double	The total cumulative time spent at this location. This can be helpful for performance monitoring.

CHAPTER 9 Core Calls

The following table lists supported calls in the API in alphabetical order, and provides a brief description for each. Click a call name to see syntax, usage, and more information for that call.

 **Note:** For a list of Apex-related calls, see [Apex-Related Calls](#), for a list of describe calls, see [Describe Calls](#), and for a list of utility calls, see [Utility Calls](#).

Call	Description
convertLead()	Converts a Lead into an Account , Contact , or (optionally) an Opportunity .
create()	Adds one or more new individual objects to your organization's data.
delete()	Deletes one or more individual objects from your organization's data.
deleteByExample()	Deletes objects from your organization's data using an sObject as a template for what to delete. All data in a big object matching the values in the sObject templates are deleted.
emptyRecycleBin()	Delete records from the recycle bin immediately.
executeListView()	Executes a list view's SOQL query to retrieve data, labels, and actions from a list view.
findDuplicates()	Performs rule-based searches for duplicate records. The input is an array of sObject , each of which specifies the values to search for and the type of object that supplies the duplicate rules. The output identifies the detected duplicates for each object that supplies the duplicate rules. <code>findDuplicates()</code> applies the rules to the values to do the search. The output identifies the detected duplicates for each sObject.
findDuplicatesByIds()	Performs rule-based searches for duplicate records. The input is an array of IDs, each of which specifies the records for which to search for duplicates. The output identifies the detected duplicates for each object that supplies the duplicate rules. <code>findDuplicatesByIds()</code> applies the rules to the record IDs to do the search. The output identifies the detected duplicates for each ID.
getDeleted()	Retrieves the IDs of individual objects of the specified object that have been deleted since the specified time. For information on IDs, see ID Field Type .
getUpdated()	Retrieves the IDs of individual objects of the specified object that have been updated since the specified time. For information on IDs, see ID Field Type .
invalidateSessions()	Ends one or more sessions specified by <code>sessionId</code> .
login()	Logs in to the login server and starts a client session.

Call	Description
<code>logout()</code>	Ends the session of the logged-in user.
<code>merge()</code>	Merges records of the same object type.
<code>performQuickActions()</code>	Executes quick actions of type create or update.
<code>process()</code>	Submits an array of approval process instances for approval, or processes an array of approval process instances to be approved, rejected, or removed.
<code>query()</code>	Executes a query against the specified object and returns data that matches the specified criteria.
<code>queryAll()</code>	Same as <code>query()</code> , but includes deleted and archived items.
<code>queryMore()</code>	Retrieves the next batch of objects from a query.
<code>retrieve()</code>	Retrieves one or more objects based on the specified object IDs.
<code>search()</code>	Executes a text search in your organization's data.
<code>undelete()</code>	Undelete records identified with <code>queryAll()</code> .
<code>update()</code>	Updates one or more existing objects in your organization's data.
<code>upsert()</code>	Creates new objects and updates existing objects; matches on a custom field to determine the presence of existing objects.

Samples

The samples in this section are based on the enterprise WSDL file. They assume that you have already imported the WSDL file and created a connection. To learn how to do so, see the [Quick Start](#) tutorial.

convertLead()

Converts a [Lead](#) into an [Account](#), [Contact](#), or (optionally) an [Opportunity](#).

Syntax

```
LeadConvertResult[] = connection.convertLead(leadConverts LeadConvert[]);
```

Usage

Use `convertLead()` to convert a [Lead](#) into an [Account](#) and [Contact](#), and (optionally) an [Opportunity](#). If appropriate for your business, you can also use `convertLead()` to convert a lead to an account and a person account instead of a contact. To convert a [Lead](#), your client application must be logged in with the "Convert Leads" permission and the "Edit" permission on leads, as well as "Create" and "Edit" on the [Account](#), [Contact](#), and [Opportunity](#) objects.

This call provides an easy way to convert the information in a qualified lead to a new or updated account, contact, and opportunity. Your organization can set its own guidelines for determining when a lead is qualified. Typically, a lead can be converted when it becomes a real opportunity that you want to forecast.

If data is merged into existing account, contact, and opportunity objects, then only empty fields in the target object are overwritten—existing data (including IDs) aren't overwritten. The only exception is if your client application sets `overwriteLeadSource` to `true`. In this case, the `LeadSource` field in the target [Contact](#) object will be overwritten with the contents of the `LeadSource` field in the source [Lead](#) object.

When converting leads, consider the following rules and guidelines:

Field Mappings

The system automatically maps standard lead fields to standard account, contact, and opportunity fields. For custom lead fields, your Salesforce administrator can specify how they map to custom account, contact, and opportunity fields.

Record Types

If the organization uses record types, the default record type of the new owner is assigned to records created during lead conversion. For more information about record types, see [Salesforce Help](#).

Picklist Values

The system assigns the default picklist values for the account, contact, and opportunity when mapping any standard lead picklist fields that are blank. If your organization uses record types, blank values are replaced with the default picklist values of the new record owner.

String Values

Starting with API version 15.0, if you specify a value for a field that contains a string, and the value is too big for the field, the call fails and an error is returned. In previous versions of the API the value was truncated and the call succeeded. If you wish to keep the old behavior with versions 15.0 and later, use the [AllowFieldTruncationHeader](#) SOAP header.

Errors

If any of the leads fail to convert as part of a bulk operation, the lead conversion is retried for each lead individually.

Automatic Subscriptions for Chatter Feeds

When you convert a lead into a new account, contact, and opportunity, the lead owner is unsubscribed from the lead record's Chatter feed. The lead owner, the owner of the generated records, and users that were subscribed to the lead aren't automatically subscribed to the generated records, unless they have automatic subscriptions enabled in their Chatter feed settings. They must have automatic subscriptions enabled to see changes to the account, contact, and opportunity records in their news feed.

A user can subscribe to a record or to another user. Changes to the record and updates from the users are displayed in the Chatter feed on the user's home page, which is a useful way to stay up-to-date with other users and with changes made to records in Salesforce. Feeds are available in API version 18.0 and later.

Basic Steps for Converting Leads

Converting leads involves the following basic steps:

1. The client application determines the IDs of any lead(s) to be converted.
2. Optionally, the client application determines the IDs of any account(s) to merge the lead into. The client application can use SOSL or SOQL to search for accounts that match the lead name, as in the following example:

```
select id, name from account where name='CompanyNameOfLeadBeingMerged'
```

3. Optionally, the client application determines the IDs of contact(s) to merge the lead into. The client application can use SOSL or SOQL to search for contacts that match the lead contact name, as in the following example:

```
select id, name from contact where firstName='FirstName' and lastName='LastName' and
accountId = '001...'
```

4. Optionally, the client application determines whether opportunities should be created from the lead, or the ID of an opportunity to merge the lead into. The client application can use SOSL or SOQL to search for contacts that match the lead contact name, as in the following example:

```
select id, name from opportunity where name='OpportunityNameOfOpportunityBeingMerged'
```

5. The client application queries the LeadStatus table to obtain the possible converted status options (

```
SELECT Id, MasterLabel FROM LeadStatus WHERE IsConverted=true
```

), and then selects a value for the Converted Status.

6. The client application calls `convertLead()`.
7. The client application iterates through the returned result and examines each `LeadConvertResult` object to determine whether conversion succeeded for each lead.
8. As an optional best practice, the client application creates tasks in which the `WhoId` is the `ContactId` and, if an opportunity is created, the `WhatId` is the `OpportunityId`.
9. Optionally, when converting leads owned by a queue, the owner must be specified. This is because accounts and contacts cannot be owned by a queue. Even if you are specifying an existing account or contact, you must still specify an owner.

Sample Code—Java

This sample shows how to convert leads. It creates two leads and converts them. Next, it iterates through the lead conversion results and writes the IDs of the account, contact, and opportunity created for each lead.

```
public String[] convertLeadRecords() {
    String[] result = new String[4];
    try {

        // Create two leads to convert
        Lead[] leads = new Lead[2];
        Lead lead = new Lead();
        lead.setLastName("Mallard");
        lead.setFirstName("Jay");
        lead.setCompany("Wingo Ducks");
        lead.setPhone("(707) 555-0328");
        leads[0] = lead;
```

```

lead = new Lead();
lead.setLastName("Platypus");
lead.setFirstName("Ogden");
lead.setCompany("Denio Water Co.");
lead.setPhone("(775) 555-1245");
leads[1] = lead;
SaveResult[] saveResults = connection.create(leads);

// Create a LeadConvert array to be used
// in the convertLead() call
LeadConvert[] leadsToConvert = new LeadConvert[saveResults.length];

for (int i = 0; i < saveResults.length; ++i) {
    if (saveResults[i].isSuccess()) {
        System.out
            .println("Created new Lead: " + saveResults[i].getId());
        leadsToConvert[i] = new LeadConvert();
        leadsToConvert[i].setConvertedStatus("Closed - Converted");
        leadsToConvert[i].setLeadId(saveResults[i].getId());
        result[0] = saveResults[i].getId();
    } else {
        System.out.println("\nError creating new Lead: "
            + saveResults[i].getErrors()[0].getMessage());
    }
}
// Convert the leads and iterate through the results
LeadConvertResult[] lcResults = connection.convertLead(leadsToConvert);
for (int j = 0; j < lcResults.length; ++j) {
    if (lcResults[j].isSuccess()) {
        System.out.println("Lead converted successfully!");
        System.out.println("Account ID: " + lcResults[j].getAccountId());
        System.out.println("Contact ID: " + lcResults[j].getContactId());
        System.out.println("Opportunity ID: "
            + lcResults[j].getOpportunityId());
    } else {
        System.out.println("\nError converting new Lead: "
            + lcResults[j].getErrors()[0].getMessage());
    }
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
return result;
}

```

Sample Code—C#

This sample shows how to convert leads. It creates two leads and converts them. Next, it iterates through the lead conversion results and writes the IDs of the account, contact, and opportunity created for each lead.

```

public String[] convertLeadRecords()
{
    String[] result = new String[4];

```

```
try
{
    // Create two leads to convert
    Lead[] leads = new Lead[2];
    Lead lead = new Lead();
    lead.LastName = "Mallard";
    lead.FirstName = "Jay";
    lead.Company = "Wingo Ducks";
    lead.Phone = "(707) 555-0328";
    leads[0] = lead;
    lead = new Lead();
    lead.LastName = "Platypus";
    lead.FirstName = "Ogden";
    lead.Company = "Denio Water Co.";
    lead.Phone = "(775) 555-1245";
    leads[1] = lead;
    SaveResult[] saveResults = binding.create(leads);

    // Create a LeadConvert array to be used
    // in the convertLead() call
    LeadConvert[] leadsToConvert =
        new LeadConvert[saveResults.Length]; ;
    for (int i = 0; i < saveResults.Length; ++i)
    {
        if (saveResults[i].success)
        {
            Console.WriteLine("Created new Lead: " +
                saveResults[i].id);
            leadsToConvert[i] = new LeadConvert();
            leadsToConvert[i].convertedStatus = "Closed - Converted";
            leadsToConvert[i].leadId = saveResults[i].id;
            result[0] = saveResults[i].id;
        }
        else
        {
            Console.WriteLine("\nError creating new Lead: " +
                saveResults[i].errors[0].message);
        }
    }
}
// Convert the leads and iterate through the results
LeadConvertResult[] lcResults =
    binding.convertLead(leadsToConvert);
for (int j = 0; j < lcResults.Length; ++j)
{
    if (lcResults[j].success)
    {
        Console.WriteLine("Lead converted successfully!");
        Console.WriteLine("Account ID: " +
            lcResults[j].accountId);
        Console.WriteLine("Contact ID: " +
            lcResults[j].contactId);
        Console.WriteLine("Opportunity ID: " +
            lcResults[j].opportunityId);
    }
}
```




```

        else
        {
            Console.WriteLine("\nError converting new Lead: " +
                lcResults[j].errors[0].message);
        }
    }
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
return result;
}

```

LeadConvert Arguments

This call accepts an array of LeadConvert objects (100 maximum). A LeadConvert object contains the following properties.

Name	Type	Description
accountId	ID	ID of the Account into which the lead will be merged. Required only when updating an existing account, including person accounts. If no <code>accountId</code> is specified, then the API creates a new account. To create a new account, the client application must be logged in with sufficient access rights. To merge a lead into an existing account, the client application must be logged in with read/write access to the specified account. The account name and other existing data are not overwritten. For information on IDs, see ID Field Type .
contactId	ID	ID of the Contact into which the lead will be merged (this contact must be associated with the specified <code>accountId</code> , and an <code>accountId</code> must be specified). Required only when updating an existing contact.  Important: If you're converting a lead into a person account , do not specify the <code>contactId</code> or an error will result. Specify only the <code>accountId</code> of the person account. If no <code>contactId</code> is specified, then the API creates a contact that is implicitly associated with the Account . To create a new contact, the client application must be logged in with sufficient access rights. To merge a lead into an existing contact, the client application must be logged in with read/write access to the specified contact. The contact name and other existing data aren't overwritten (unless <code>overwriteLeadSource</code> is set to <code>true</code> , in which case only the <code>LeadSource</code> field is overwritten).
convertedStatus	string	Valid LeadStatus value for a converted lead. Required. To obtain the list of possible values, the client application queries the LeadStatus object. For example: <pre>SELECT Id, MasterLabel FROM LeadStatus WHERE IsConverted=true</pre>

Name	Type	Description
doNotCreateOpportunity	boolean	Specifies whether to create an Opportunity during lead conversion (<code>false</code> , the default) or not (<code>true</code>). Set this flag to <code>true</code> only if you don't want to create an opportunity from the lead. An opportunity is created by default.
leadId	ID	ID of the Lead to convert. Required. For information on IDs, see ID Field Type .
opportunityId	ID	The ID of an existing opportunity to relate to the lead. The <code>opportunityId</code> and <code>opportunityName</code> arguments are mutually exclusive. Specifying a value for both results in an error. If <code>doNotCreateOpportunity</code> argument is <code>true</code> , then no Opportunity is created and this field must be left blank; otherwise, an error is returned.
opportunityName	string	Name of the opportunity to create. If no name is specified, then this value defaults to the company name of the lead. The maximum length of this field is 80 characters. The <code>opportunityId</code> and <code>opportunityName</code> arguments are mutually exclusive. Specifying a value for both results in an error. If <code>doNotCreateOpportunity</code> argument is <code>true</code> , then no Opportunity is created and this field must be left blank; otherwise, an error is returned.
overwriteLeadSource	boolean	Specifies whether to overwrite the <code>LeadSource</code> field on the target Contact object with the contents of the <code>LeadSource</code> field in the source Lead object (<code>true</code>), or not (<code>false</code> , the default). To set this field to <code>true</code> , the client application must specify a <code>contactId</code> for the target contact.
ownerId	ID	Specifies the ID of the person to own any newly created account, contact, and opportunity. If the client application doesn't specify this value, then the owner of the new object will be the owner of the lead. Not applicable when merging with existing objects—if an <code>ownerId</code> is specified, the API doesn't overwrite the <code>ownerId</code> field in an existing account or contact. For information on IDs, see ID Field Type .
relatedPersonAccountId	ID	When converting a lead to a business account and a person account instead of a contact, specifies the ID of the <i>existing</i> person account to convert the lead to.
relatedPersonAccountRecord	Entity	When converting a lead to a business account and a person account instead of a contact, specifies the entity record of the <i>new</i> person account to convert the lead to.
sendNotificationEmail	boolean	Specifies whether to send a notification email to the owner specified in the <code>ownerId</code> (<code>true</code>) or not (<code>false</code> , the default).

Response

[LeadConvertResult\[\]](#)

Fault

[UnexpectedErrorFault](#)

SEE ALSO:

[API Call Basics](#)

LeadConvertResult

This call returns an array of `LeadConvertResult` objects. Each element in the `LeadConvertResult` array corresponds to the `LeadConvert[]` array passed as the `leadConverts` parameter in the `convertLead()` call. For example, the object returned in the first index in the `LeadConvertResult` array matches the object specified in the first index of the `LeadConvert[]` array. A `LeadConvertResult` object has the following properties:

Name	Type	Description
<code>accountId</code>	ID	ID of the new Account (if a new account was specified) or the ID of the account specified when <code>convertLead()</code> was invoked.
<code>contactId</code>	ID	ID of the new Contact (if a new contact was specified) or the ID of the contact specified when <code>convertLead()</code> was invoked. For information on IDs, see ID Field Type .
<code>leadId</code>	ID	ID of the converted Lead . For information on IDs, see ID Field Type .
<code>opportunityId</code>	ID	ID of the new or existing Opportunity , if one was created or related to the lead when <code>convertLead()</code> was invoked. For information on IDs, see ID Field Type .
<code>relatedPersonAccountId</code>	ID	ID of the new or existing related Person Account , if one was created or related to the lead when <code>convertLead()</code> was invoked. For information on IDs, see ID Field Type .
<code>success</code>	boolean	Indicates whether the <code>convertLead()</code> call succeeded (<code>true</code>) or not (<code>false</code>) for this object.
<code>errors</code>	Error[]	If an error occurred during the <code>create()</code> call, an array of one or more Error objects providing the error code and description.

create()



[other]: Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

Adds one or more new records to your organization's data.

Syntax

```
SaveResult[] = connection.create(sObject[] sObjects);
```

Usage

Use `create()` to add one or more records, such as an [Account](#) or [Contact](#) record, to your organization's information. The `create()` call is analogous to the INSERT statement in SQL.

When creating objects, consider the following rules and guidelines.

Permissions

Your client application must be logged in with sufficient access rights to create records within the specified object. For more information, see [Factors that Affect Data Access](#).

Special Handling

Certain objects—and certain fields within those objects—require special handling or permissions. For example, you might also need permissions to access the object's parent object. Before you attempt to `create()` a record for a particular object, be sure to read its description in the [Standard Objects](#).

Createable Fields

Only objects where `createable` is `true` can be created via the `create()` call. To determine whether a given object can be created, your client application can invoke the `describeSObjects()` [describeSObjects\(\)](#) call on the object and inspect its `createable` property.

Automatically Maintained Fields

The API generates unique values for ID fields automatically. For `create()`, you cannot explicitly specify an ID value in the `sObject`. The `saveResult[]` object contains the ID of each record that was successfully created. For information on IDs, see [ID Field Type](#).

The API populates certain fields automatically, such as `CreatedDate`, `CreatedById`, `LastModifiedDate`, `LastModifiedById`, and `SystemModstamp`. You cannot explicitly specify these values.

Required Fields

For required fields that do not have a preconfigured default value, you must supply a value. For more information, see [Required Fields](#).

Default Values

For some objects, some fields have a default value, such as `OwnerId`. If you do not specify a value for such fields, the API populates the fields with the default value. For example, if you do not override `OwnerId`, then the API populates this field with the user ID associated with the user as whom your client application is logged in.

- For required fields that do not have a preconfigured default value, you must supply a value.
- For all other fields in the object, if you do not explicitly specify a value, then its value is `null` (`VT_EMPTY`).

Referential Integrity

Your client application must conform to the rules of referential integrity. For example, if you are creating a record for an object that is the child of a parent object, you must supply the foreign key information that links the child to the parent. For example, when creating a [CaseComment](#), you must supply the valid case ID for the parent [Case](#), and that parent Case must exist in the database.

Valid Data Values

You must supply values that are valid for the field's data type, such as integers (not alphabetic characters) for integer fields. In your client application, follow the data formatting rules specified for your programming language and development tool (your development tool will handle the appropriate mapping of data types in SOAP messages).

String Values

When storing values in string fields, the API trims any leading and trailing whitespace. For example, if the value of a name field is entered as " ABC Company ", then the value is stored in the database as "ABC Company".

Starting with API version 15.0, if you specify a value for a field that contains a string, and the value is too big for the field, the call fails and an error is returned. In previous versions of the API the value was truncated and the call succeeded. If you wish to keep the old behavior with versions 15.0 and later, use the [AllowFieldTruncationHeader](#) SOAP header.

Assignment Rules

When creating new [Account](#) (accounts fire Territory Management assignment rules), [Case](#), or [Lead](#) records, your client application can set options in the [AssignmentRuleHeader](#) to have the case or lead automatically assigned to one or more users based on assignment rules configured in the Salesforce user interface.

Maximum Number of Records Created

Your client application can add up to 200 records in a single `create()` call. If a create request exceeds 200 records, then the entire operation fails.

Rollback on Error

The [AllOrNoneHeader](#) header allows you to roll back all changes unless all records are processed successfully. This header is available in API version 20.0 and later. Allows a call to roll back all changes unless all records are processed successfully.

Automatic Subscriptions for Chatter Feeds

To subscribe to records they create, users must enable the `Automatically follow records that I create` option in their personal settings. If users have automatic subscriptions enabled, they automatically follow the records they create and see changes to those records in their Chatter feed on the Home tab.

A user can subscribe to a record or to another user. Changes to the record and updates from the users are displayed in the Chatter feed on the user's home page, which is a useful way to stay up-to-date with other users and with changes made to records in Salesforce. Feeds are available in API version 18.0 and later. The `EntitySubscription` object represents a subscription of a user following a record or another user.

Disabling Feed Notifications

If you're processing a large number of records and don't want to track the changes in various feeds related to the records, use [DisableFeedTrackingHeader](#). This is especially useful for bulk changes.

Creating Records for Different Object Types

You can create records for multiple object types, including custom objects, in one call with API version 20.0 and later. For example, you could create a contact and an account in one call. You can create records for up to 10 object types in one call.

Records are saved in the same order that they are entered in the `sObjects` input array. If you are entering new records that have a parent-child relationship, the parent record must precede the child record in the `sObjects` array. For example, if you are creating a contact that references an account that is also being created in the same call, the account must have a smaller index in the `sObjects` array than the contact does. The contact references the account by using an `External ID` field.

You can't add a record that references another record of the same object type in the same call. For example, the Contact object has a `Reports To` field that is a reference to another contact. You can't create two contacts in one call if one contact uses the `Reports To` field to reference a second contact in the `sObjects` array. You can create a contact that references another contact that has been previously created.


Records for different object types are broken into multiple chunks by Salesforce. A chunk is a subset of the `sObjects` input array and each chunk contains records of one object type. Data is committed on a chunk-by-chunk basis. Any Apex triggers related to the records in a chunk are invoked once per chunk. Consider an `sObjects` input array containing the following set of records:

```
account1, account2, contact1, contact2, contact3, case1, account3, account4, contact4
```

Salesforce splits the records into five chunks:

1. `account1, account2`
2. `contact1, contact2, contact3`
3. `case1`
4. `account3, account4`
5. `contact4`

Each call can process up to 10 chunks. If the `sObjects` array contains more than 10 chunks, you must process the records in more than one call.

 **Warning:** You can't create records for multiple object types in one call if one of those types is related to a feature in the Setup area in Salesforce. The only exceptions are the following objects:

- Custom settings objects, which are similar to custom objects. For more information, see “Create Custom Settings” in Salesforce Help.
- GroupMember
- Group
- User if the `UserRoleId` field is not being set.

`create()` and Foreign Keys

You can use external ID fields as a foreign key, which allows you to create a record and relate it to another existing record in a single step instead of querying the parent record ID first. To do this, set the foreign key field to an instance of the parent `sObject` that only has the external ID field specified. This external ID should match the external ID value on the parent record.

The following Java and C# examples show you how to create an opportunity and relate it to an existing account using a custom external ID field named `MyExtId__c`. Each example creates an opportunity, sets the required fields, and then sets the opportunity external ID field to the account object that has only the external ID field specified. The code then creates the opportunity. Once the opportunity is created, the account will be its parent.

Java Example

```
public void createForeignKeySample() {
    try {
        Opportunity newOpportunity = new Opportunity();
        newOpportunity.setName("OpportunityWithFK");
        newOpportunity.setStageName("Prospecting");
        Calendar dt = connection.getServerTimestamp().getTimestamp();
        dt.add(Calendar.DAY_OF_MONTH, 7);
        newOpportunity.setCloseDate(dt);

        Account parentAccountRef = new Account();
        parentAccountRef.setMyExtId__c("SAP1111111");
        newOpportunity.setAccount(parentAccountRef);

        SaveResult[] results = connection
            .create(new SObject[] { newOpportunity });
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

C# Example

```
public void createForeignKeySample()
{
    try
    {
        Opportunity newOpportunity = new Opportunity();
        newOpportunity.Name = "OpportunityWithFK";
        newOpportunity.StageName = "Prospecting";
        DateTime dt = (DateTime)binding.getServerTimestamp().timestamp;
        newOpportunity.CloseDate = dt.AddDays(7);
        newOpportunity.CloseDateSpecified = true;

        // Create the parent reference.
        // Used only for foreign key reference
        // and doesn't contain any other fields
        Account accountReference = new Account();
        accountReference.MyExtId__c = "SAP1111111";
        newOpportunity.Account = accountReference;

        // Create the account and the opportunity
        SaveResult[] results = binding.create(new sObject[] {
            newOpportunity });
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

```

    }
}

```

Creating Parent and Child Records in a Single Call 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 call 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 `create()` call. This sObject must have the required fields (and optionally other fields) set in addition to the external ID field.
- Call `create()` 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.

The parent and child records are records related through a predefined relationship, such as a master-detail or lookup relationship. 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](#).

The following Java and C# examples show you how to create an opportunity with a parent account in the same `create()` call. Each example creates an Opportunity sObject and populates some of its fields, then creates two Account objects. The first account is only for the foreign key relationship, and the second is for the account creation and has the account fields set. Both accounts have the external ID field, `MyExtID__c`, set. Next, the sample calls `create()` by passing it an array of sObjects. The first element in the array is the parent sObject and the second is the opportunity sObject. The `create()` call creates the opportunity with its parent account in a single call. Finally, the sample checks the results of the call and writes the IDs of the created records to the console, or the first error if record creation fails.

Java Example

```

public void createForeignKeySample() {
    try {
        Opportunity newOpportunity = new Opportunity();
        newOpportunity.setName("OpportunityWithAccountInsert");
        newOpportunity.setStageName("Prospecting");
        Calendar dt = connection.getServerTimestamp().getTimestamp();
        dt.add(Calendar.DAY_OF_MONTH, 7);
        newOpportunity.setCloseDate(dt);

        // Create the parent reference.
        // Used only for foreign key reference
        // and doesn't contain any other fields.
        Account accountReference = new Account();
        accountReference.setMyExtID__c("SAP111111");
        newOpportunity.setAccount(accountReference);

        // Create the Account object to insert.
        // Same as above but has Name field.
        // Used for the create call.
        Account parentAccount = new Account();
        parentAccount.setName("Hallie");
        parentAccount.setMyExtID__c("SAP111111");
    }
}

```



```

// Create the account and the opportunity.
SaveResult[] results = connection.create(new SObject[] {
    parentAccount, newOpportunity });

// Check results.
for (int i = 0; i < results.length; i++) {
    if (results[i].isSuccess()) {
        System.out.println("Successfully created ID: "
            + results[i].getId());
    } else {
        System.out.println("Error: could not create subject "
            + "for array element " + i + ".");
        System.out.println("    The error reported was: "
            + results[i].getErrors()[0].getMessage() + "\n");
    }
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

C# Example

```

public void createForeignKeySample()
{
    try
    {
        Opportunity newOpportunity = new Opportunity();
        newOpportunity.Name = "OpportunityWithAccountInsert";
        newOpportunity.StageName = "Prospecting";
        DateTime dt = (DateTime)binding.getServerTimestamp().timestamp;
        newOpportunity.CloseDate = dt.AddDays(7);
        newOpportunity.CloseDateSpecified = true;

        // Create the parent reference.
        // Used only for foreign key reference
        // and doesn't contain any other fields.
        Account accountReference = new Account();
        accountReference.MyExtID__c = "SAP111111";
        newOpportunity.Account = accountReference;

        // Create the Account object to insert.
        // Same as above but has Name field.
        // Used for the create call.
        Account parentAccount = new Account();
        parentAccount.Name = "Hallie";
        parentAccount.MyExtID__c = "SAP111111";

        // Create the account and the opportunity.
        SaveResult[] results = binding.create(new sObject[] {
            parentAccount, newOpportunity });

        // Check results.
        for (int i = 0; i < results.Length; i++)
    }
}

```

```

    {
        if (results[i].success)
        {
            Console.WriteLine("Successfully created ID: "
                + results[i].id);
        }
        else
        {
            Console.WriteLine("Error: could not create subject "
                + "for array element " + i + ".");
            Console.WriteLine("    The error reported was: "
                + results[i].errors[0].message + "\n");
        }
    }
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

Basic Steps for Creating Records

Creating records involves the following basic steps:

1. Create an `sObject` for one or more objects. For each record, populate its fields with the data that you want to add.
2. Construct an `sObject[]` array and populate that array with the objects that you want to create.
3. Call `create()`, passing in the `sObject[]` array.
4. Process the results in the `saveResult[]` object to verify whether the records have been successfully created.

Sample Code—Java

This sample shows how to create records. It creates two `Account` objects and sets their fields. The Name of the second account isn't set so that an error occurs on creation, since Name is a required field. After making the `create()` call by passing the array containing the two accounts, the sample iterates over the results and writes the ID of the new account or an error message if the account creation fails. Finally, the sample returns an array of the new account IDs, which in this case contains only one ID.

```

public String[] createRecords() {
    // Create two accounts
    String[] result = new String[2];
    Account account1 = new Account();
    Account account2 = new Account();

    // Set some fields on the account object
    account1.setName("The Brick Hut");
    account1.setBillingStreet("403 McAdoo St");
    account1.setBillingCity("Truth or Consequences");
    account1.setBillingState("NM");
    account1.setBillingPostalCode("87901");
    account1.setBillingCountry("US");
}

```

```

// Required Name field is not being set on account2,
// so this record should fail during create.
// account2.setName("Camp One Creations");
account2.setBillingStreet("25800 Arnold Dr");
account2.setBillingCity("Sonoma");
account2.setBillingState("CA");
account2.setBillingPostalCode("95476");
account2.setBillingCountry("US");
Account[] accounts = { account1, account2 };

try {
    // Call create() to add the accounts
    SaveResult[] saveResults = connection.create(accounts);
    // Iterate through the results.
    // There should be one successful creation
    // and one failed creation.
    for (int i = 0; i < saveResults.length; i++) {
        if (saveResults[i].isSuccess()) {
            System.out.println("Successfully created Account ID: "
                + saveResults[i].getId());
            result[i] = saveResults[i].getId();
        } else {
            System.out.println("Error: could not create Account "
                + "for array element " + i + ".");
            System.out.println("    The error reported was: "
                + saveResults[i].getErrors()[0].getMessage() + "\n");
            result[i] = saveResults[i].getId();
        }
    }
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
return result;
}

```

Sample Code—C#

This sample shows how to create records. It creates two Account objects and sets their fields. The Name of the second account isn't set so that an error occurs on creation, since Name is a required field. After making the `create()` call by passing the array containing the two accounts, the sample iterates over the results and writes the ID of the new account or an error message if the account creation fails. Finally, the sample returns an array of the new account IDs, which in this case contains only one ID.

```

public String[] createRecords()
{
    // Create two accounts
    String[] result = new String[2];
    Account account1 = new Account();
    Account account2 = new Account();

    // Set some fields on the account object
    account1.Name = "The Brick Hut";
    account1.BillingStreet = "403 McAdoo St";
    account1.BillingCity = "Truth or Consequences";
}

```

```
account1.BillingState = "NM";
account1.BillingPostalCode = "87901";
account1.BillingCountry = "US";
// Required Name field is not being set on account2,
// so this record should fail during create.
// account2.Name = "Camp One Creations";
account2.BillingStreet = "25800 Arnold Dr";
account2.BillingCity = "Sonoma";
account2.BillingState = "CA";
account2.BillingPostalCode = "95476";
account2.BillingCountry = "US";
Account[] accounts = { account1, account2 };

try
{
    // Call create() to add the accounts
    SaveResult[] saveResults = binding.create(accounts);
    // Iterate through the results.
    // There should be one successful creation
    // and one failed creation.
    for (int i = 0; i < saveResults.Length; i++)
    {
        if (saveResults[i].success)
        {
            Console.WriteLine("Successfully created Account ID: " +
                saveResults[i].id);
            result[i] = saveResults[i].id;
        }
        else
        {
            Console.WriteLine("Error: could not create Account " +
                "for array element " + i + ".");
            Console.WriteLine("    The error reported was: " +
                saveResults[i].errors[0].message + "\n");
            result[i] = saveResults[i].id;
        }
    }
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}

return result;
}
```

Arguments

Name	Type	Description
<code>sObjects</code>	<code>sObject[]</code>	Array of one or more <code>sObject</code> objects to <code>create()</code> . Limit: 200 <code>sObject</code> values.

Response

`saveResult[]`

Faults

[InvalidObjectFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[upsert\(\)](#)

[API Call Basics](#)

SaveResult

The `create()` call returns an array of `SaveResult` objects. Each element in the `SaveResult` array corresponds to the `sObject[]` array passed as the `sObjects` parameter in the `create()` call. For example, the object returned in the first index in the `SaveResult` array matches the object specified in the first index of the `sObject[]` array. A `SaveResult` object has the following properties:

Name	Type	Description
<code>id</code>	ID	ID of the <code>sObject</code> that you attempted to <code>create()</code> . If this field contains a value, then the object was created successfully. If this field is empty, then the object was not created and the API returned error information instead.
<code>success</code>	boolean	Indicates whether the <code>create()</code> call succeeded (<code>true</code>) or not (<code>false</code>) for this object.
<code>errors</code>	<code>Error[]</code>	If an error occurred during the <code>create()</code> call, an array of one or more <code>Error</code> objects providing the error code and description. If your organization has active duplicate rules and a duplicate is detected, the <code>SaveResult</code> includes an <code>Error</code> with a data type of <code>DuplicateError</code> .

`delete()`

Deletes one or more records from your organization's data.

Syntax

```
DeleteResult[] = connection.delete(ID[] ids);
```

Usage

Use `delete()` to delete one or more existing records, such as individual accounts or contacts, in your organization's data. The `delete()` call is analogous to the `DELETE` statement in SQL.

Rules and Guidelines

When deleting objects, consider the following rules and guidelines:

- Your client application must be logged in with sufficient access rights to delete individual objects within the specified object. For more information, see [Factors that Affect Data Access](#).
- In addition, you might also need permission to access this object's parent object. For special access requirements, see the object's description in [Standard Objects](#).
- To ensure referential integrity, the `delete()` call supports cascading deletions. If you delete a parent object, you delete its children automatically, as long as each child object can be deleted. For example, if you delete a [Case](#), the API automatically deletes any [CaseComment](#), [CaseHistory](#), and [CaseSolution](#) objects associated with that case. However, if a [CaseComment](#) is not deletable or is currently being used, then the `delete()` call on the parent [Case](#) will fail.
- Certain objects cannot be deleted via the API. To delete an object via the `delete()` call, its object must be configured as deletable (`deletable` is `true`). To determine whether a given object can be deleted, your client application can invoke the `describeObjects()` call on the object and inspect its `deletable` property.
- You can't delete records for multiple object types in one call if one of those types is related to a feature in the Setup area in Salesforce. The only exceptions are the following objects:
 - Custom settings objects, which are similar to custom objects. For more information, see “Create Custom Settings” in Salesforce Help.
 - GroupMember
 - Group
 - User

Rollback on Error

The [AllOrNoneHeader](#) header allows you to roll back all changes unless all records are processed successfully. This header is available in API version 20.0 and later. Allows a call to roll back all changes unless all records are processed successfully.

Basic Steps for Deleting Records

Deleting records involves the following basic steps:

1. Determine the ID of each record that you want to delete. For example, you might call `query()` to retrieve a set of records that you want to delete based on specific criteria.
2. Construct an `ID[]` array and populate it with the IDs of each record that you want to delete. You can specify the IDs of different types of objects in the same call. For example, you could specify the ID for an individual [Account](#) and an individual [Contact](#) in the same array. For information on IDs, see [ID Field Type](#).

3. Call `delete()`, passing in the `ID[]` array.
4. Process the results in the `DeleteResult[]` to verify whether the records have been successfully deleted.

Sample Code—Java

This sample shows how to delete records based on record IDs. The method in this sample accepts an array of IDs, which it passes to the `delete()` call and makes the call. It then parses the results and writes the IDs of the deleted records to the console or the first returned error if the deletion failed.

```
public void deleteRecords(String[] ids) {
    try {
        DeleteResult[] deleteResults = connection.delete(ids);
        for (int i = 0; i < deleteResults.length; i++) {
            DeleteResult deleteResult = deleteResults[i];
            if (deleteResult.isSuccess()) {
                System.out
                    .println("Deleted Record ID: " + deleteResult.getId());
            } else {
                // Handle the errors.
                // We just print the first error out for sample purposes.
                Error[] errors = deleteResult.getErrors();
                if (errors.length > 0) {
                    System.out.println("Error: could not delete " + "Record ID "
                        + deleteResult.getId() + ".");
                    System.out.println("    The error reported was: ("
                        + errors[0].getStatusCode() + ") "
                        + errors[0].getMessage() + "\n");
                }
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample shows how to delete records based on record IDs. The method in this sample accepts an array of IDs, which it passes to the `delete()` call and makes the call. It then parses the results and writes the IDs of the deleted records to the console or the first returned error if the deletion failed.

```
public void deleteRecords(String[] ids)
{
    try
    {
        DeleteResult[] deleteResults = binding.delete(ids);
        for (int i = 0; i < deleteResults.Length; i++)
        {
            DeleteResult deleteResult = deleteResults[i];
            if (deleteResult.success)
            {
                Console.WriteLine("Deleted Record ID: " + deleteResult.id);
            }
        }
    }
}
```

```

    }
    else
    {
        // Handle the errors.
        // We just print the first error out for sample purposes.
        Error[] errors = deleteResult.errors;
        if (errors.Length > 0)
        {
            Console.WriteLine("Error: could not delete " + "Record ID "
                + deleteResult.id + ".");
            Console.WriteLine("    The error reported was: ("
                + errors[0].statusCode + ") "
                + errors[0].message + "\n");
        }
    }
}
}
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

Arguments

Name	Type	Description
ids	ID[]	Array of one or more IDs associated with the objects to delete. In version 7.0 and later, you can pass a maximum of 200 object IDs to the <code>delete()</code> call. In version 6.0 and earlier, the limit is 2,000.

Response

`DeleteResult[]`

Faults

`InvalidObjectFault`

`UnexpectedErrorFault`

SEE ALSO:

[API Call Basics](#)

DeleteResult

The `delete()` call returns an array of `DeleteResult` objects. Each element in the `DeleteResult` array corresponds to the `ID[]` array passed as the `ids` parameter in the `delete()` call. For example, the object returned in the first index in the `DeleteResult` array matches the object specified in the first index of the `ID[]` array.

A `DeleteResult` object has the following properties:

Name	Type	Description
<code>id</code>	ID	ID of an sObject that you attempted to delete. For information on IDs, see ID Field Type .
<code>success</code>	boolean	Indicates whether the <code>delete()</code> call succeeded (<code>true</code>) or not (<code>false</code>) for this object.
<code>errors</code>	Error[]	If an error occurred during the <code>delete()</code> call, an array of one or more Error objects providing the error information.

deleteByExample()

Use `deleteByExample()` to delete big object data from your org using an `sObject` as a template for what to delete. All data in a big object matching the values in the `sObject` templates are deleted.

Syntax

```
DeleteByExampleResult[] = connection.deleteByExample(sObject[] sObjects);
```

Rules and Guidelines

When deleting data, consider the following rules and guidelines:


- Your client application must be logged in with sufficient access rights to delete individual objects within the specified object. For more information, see [Factors that Affect Data Access](#).
- You can't delete records for multiple object types in one call if one of those types is related to a feature in the Setup area in Salesforce. The only exceptions are the following objects:
 - Custom settings objects, which are similar to custom objects. For more information, see “Create Custom Settings” in the Salesforce Help.
 - GroupMember
 - Group
 - User

Basic Steps for Deleting Data

Deleting data involves the following basic steps:

1. Define an `sObject` using all the fields that make up the index of the big object.
2. Specify the values for each field.
3. Call `deleteByExample()`, passing in the `sObject` you created.

4. Process the results in the `DeleteByExampleResult []` to verify whether the records have been successfully deleted.

 **Note:** Repeating a successful `deleteByExample()` operation results in success, even if the data has already been deleted.

Sample Code—Custom Big Objects

This sample shows how to delete records in a custom big object. In this example, `Account__c`, `Game_Platform__c`, and `Play_Date__c` are part of the custom big object's index. All rows where `Account__c` is "001d000000Ky3xIAB", `Game_Platform__c` is "iOS", and `Play_Date__c` is "2017-11-28T19:13:36.000z" are deleted.

```
public static void main(String[] args) {
    try{
        //Declare an sObject that has the values to delete
        sObject[] sObjectsToDelete = new sObject[1];
        sObject[] customerBO = new sObject();
        customerBO.setType("Customer_Interaction__b");
        customerBO.setField("Account__c", "001d000000Ky3xIAB");
        customerBO.setField("Game_Platform__c", "iOS");
        customerBO.setField("Play_Date__c", "2017-11-28T19:13:36.000z");
        sObjectsToDelete[0] = customerBO;

        DeleteByExampleResult[] result = connection.deleteByExample(sObjectsToDelete);
    }
}
```

Sample Code—Field Audit Trail

This sample shows how to delete records in `FieldHistoryArchive`. All rows with the specified criteria are deleted.

```
public static void main(String[] args) {
    try{
        //Declare an sObject that has the values to delete
        sObject[] sObjectsToDelete = new sObject[2];
        sObject[] fieldHistoryArchive_1 = new sObject();
        fieldHistoryArchive_1.setType("FieldHistoryArchive");
        fieldHistoryArchive_1.setField("FieldHistoryType", "Account");
        fieldHistoryArchive_1.setField("ParentId", "001d000000Ky3xIAB");
        fieldHistoryArchive_1.setField("CreatedDate", "2017-11-28T19:13:36.000z");
        fieldHistoryArchive_1.setField("HistoryId", "017D000000ESURXIA5");
        sObjectsToDelete[0] = fieldHistoryArchive_1;

        sObject[] fieldHistoryArchive_2 = new sObject();
        fieldHistoryArchive_2.setType("FieldHistoryArchive");
        fieldHistoryArchive_2.setField("FieldHistoryType", "Account");
        fieldHistoryArchive_2.setField("ParentId", "001d000000Ky3xIAB");
        fieldHistoryArchive_2.setField("CreatedDate", "2017-11-29T19:13:36.000z");
        fieldHistoryArchive_2.setField("HistoryId", "017D000000ESURMIA5");
        sObjectsToDelete[1] = fieldHistoryArchive_2;

        DeleteByExampleResult[] result = connection.deleteByExample(sObjectsToDelete);
    }
}
```

Arguments

Name	Type	Description
sObject	sObject[]	Array of one or more sObjects to use as templates for deletion.

Response

[DeleteByExampleResult](#)[]

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

DeleteByExampleResult

The `deleteByExample()` call returns an array of `DeleteByExampleResult` objects. Each element in the `DeleteByExampleResult` array corresponds to the `sObject[]` array passed in the `deleteByExample()` call. For example, the object returned in the first index in the `DeleteByExampleResult` array matches the `sObject` specified in the first index of the `sObject[]` array.

A `DeleteByExampleResult` object has the following properties:

Name	Type	Description
entity	sObject	Details for the <code>sObject</code> that you attempted to delete.
rowCount	long	Indicates the number of rows that were deleted.
success	boolean	Indicates whether the <code>deleteByExample()</code> call succeeded (<code>true</code>) or not (<code>false</code>) for this object.
errors	Error[]	If an error occurred during the <code>deleteByExample()</code> call, an array of one or more Error objects providing the error information.

emptyRecycleBin()

Delete records from the recycle bin immediately.

Syntax

```
EmptyRecycleBinResult[] = connection.emptyRecycleBin(ID[] ids);
```

Usage

The Recycle Bin lets you view and restore recently deleted records for 15 days before they are permanently deleted. Your org can have up to 5,000 records per license in the Recycle Bin at any one time. For example, if your org has five user licenses, 25,000 records can be stored in the Recycle Bin. If your org reaches its Recycle Bin limit, Salesforce automatically removes the oldest records, as long as they have been in the recycle bin for at least two hours.

If you know you will be adding a great number of records to the Recycle Bin and you know you won't need to `undelete()` them, you may wish to remove them before the Salesforce process deletes records. For example, you can use this call if you are loading a large number of records for testing, or if you are doing a large number of `create()` calls followed by `delete()` calls.

Rules and Guidelines

When emptying recycle bins, consider the following rules and guidelines:

- 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.
- Available in version 10.0 and later.
- Maximum number of records is 200.
- Do not include the IDs of any records that will be cascade deleted, or an error will occur.
- Once records are deleted using this call, they cannot be `undelete()`.
- After records are deleted from the Recycle Bin using this call, they can be queried using `queryAll()` for some time. Typically this time is 24 hours, but may be shorter or longer.

Sample Code—Java

This sample shows how to empty the Recycle Bin. It accepts an array containing the IDs of the records to remove from the Recycle Bin. It calls `emptyRecycleBin()` and passes it the array of IDs. Next, it iterates over the results and writes the IDs of the removed records or the first error of the failed records to the console.

```
public void emptyRecycleBin(String[] ids) {
    try {
        EmptyRecycleBinResult[] emptyRecycleBinResults = connection
            .emptyRecycleBin(ids);
        for (int i = 0; i < emptyRecycleBinResults.length; i++) {
            EmptyRecycleBinResult emptyRecycleBinResult = emptyRecycleBinResults[i];
            if (emptyRecycleBinResult.isSuccess()) {
                System.out.println("Recycled ID: "
                    + emptyRecycleBinResult.getId());
            } else {
                Error[] errors = emptyRecycleBinResult.getErrors();
                if (errors.length > 0) {
                    System.out
                        .println("Error code: " + errors[0].getStatusCode());
                    System.out
                        .println("Error message: " + errors[0].getMessage());
                }
            }
        }
    }
    catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

```
}  
}
```

Sample Code—C#

This sample shows how to empty the Recycle Bin. It accepts an array containing the IDs of the records to remove from the Recycle Bin. It calls `emptyRecycleBin()` and passes it the array of IDs. Next, it iterates over the results and writes the IDs of the removed records or the first error of the failed records to the console.

```
public void emptyRecycleBin(String[] ids)  
{  
    try  
    {  
        EmptyRecycleBinResult[] emptyRecycleBinResults =  
            binding.emptyRecycleBin(ids);  
        for (int i = 0; i < emptyRecycleBinResults.Length; i++)  
        {  
            EmptyRecycleBinResult emptyRecycleBinResult = emptyRecycleBinResults[i];  
            if (emptyRecycleBinResult.success)  
            {  
                Console.WriteLine("Recycled ID: "  
                    + emptyRecycleBinResult.id);  
            }  
            else  
            {  
                Error[] errors = emptyRecycleBinResult.errors;  
                if (errors.Length > 0)  
                {  
                    Console.WriteLine("Error code: " + errors[0].statusCode);  
                    Console.WriteLine("Error message: " + errors[0].message);  
                }  
            }  
        }  
    }  
    catch (SoapException e)  
    {  
        Console.WriteLine("An unexpected error has occurred: " +  
            e.Message + "\n" + e.StackTrace);  
    }  
}
```

Arguments

Name	Type	Description
ids	ID[]	Array of one or more IDs associated with the records to delete from the Recycle Bin. Maximum number of records is 200.

Response

[EmptyRecycleBinResult](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[delete\(\)](#)

[undelete\(\)](#)

EmptyRecycleBinResult

The `emptyRecycleBin()` call returns an array of `EmptyRecycleBinResult` objects. Each element in the array corresponds to an element in the `ID[]` array passed as the parameter in the `emptyRecycleBin()` call. For example, the object returned in the first index in the `EmptyRecycleBinResult` array matches the object specified in the first index of the `ID[]` array.

A `EmptyRecycleBinResult` object has the following properties:

Name	Type	Description
<code>id</code>	ID	ID of an sObject that you attempted to delete from the Recycle Bin. For information about IDs, see ID Field Type .
<code>isSuccess</code>	boolean	Indicates whether the call succeeded (<code>true</code>) or not (<code>false</code>) for this record.
<code>errors</code>	Error[]	If an error occurred during the call, an array of one or more Error objects providing the error information.

`executeListView()`

Executes a list view's SOQL query to retrieve data, labels, and actions from a list view.

Syntax

```
ExecuteListViewResult result = connection.executeListView(ExecuteListViewRequest request);
```

Usage

The `executeListView()` call takes an [ExecuteListViewRequest](#) object, executes the SOQL query for the list view, and returns the resulting data and presentation information in an [ExecuteListViewResult](#) object. This call is available in API version 32.0 and later.

Sample Code—Java

```
private void example(ApiProtocol protocol, AppVersion version) throws Exception {
    // Get the list results via the list view API
    EnterpriseConnection connection =
makeClient(getUserUtil().getUserWithModifyAllData(), AppVersion.VERSION_190,
            getName());
    ExecuteListViewRequest request = new ExecuteListViewRequest();
    request.setObjectType("Account");
    request.setDeveloperNameOrId(listViews[0].getId());
    request.setLimit(50000);

    com.sforce.soap.enterprise.ExecuteListViewResult result =
connection.executeListView(request);
}
```

Arguments

Name	Type	Description
request	ExecuteListViewRequest	An object that specifies the list view and the limit, offset, and ordering of the results.

Response

An [ExecuteListViewResult](#) object.

ExecuteListViewRequest

Use the [ExecuteListViewRequest](#) object with `executeListView()` to retrieve data, labels, and actions from a list view.

The [ExecuteListViewRequest](#) object has the following properties:

Name	Type	Description
developerNameOrId	string	The list view's ID or fully qualified developer name.
limit	int	The maximum number of records to return. Default: 25
offset	int	The number of records to skip. Default: 0
orderBy	ListViewOrderBy[]	The order in which to return the records.
subjectType	string	The API name of the sObject for the list view.

ExecuteListViewResult

Contains list view data that you retrieve programmatically.

To retrieve an `executeListViewResult` object, use the `executeListView()` call. The `executeListViewResult` object has the following properties:

Name	Type	Description
<code>columns</code>	<code>ListViewColumn[]</code>	An array of the columns in the list view.
<code>developerName</code>	<code>string</code>	The list view's fully qualified developer name.
<code>done</code>	<code>boolean</code>	If <code>true</code> , indicates that all records have been returned.
<code>id</code>	<code>ID</code>	The list view's ID.
<code>label</code>	<code>string</code>	The display label of the list view.
<code>records</code>	<code>ListViewRecord[]</code>	An array of records that match the list view query.
<code>size</code>	<code>int</code>	The number of records that are returned by the list view query.

ListViewColumn

Contains metadata about a single list view column.

The `ListViewColumn` object is returned by the `describeSoqlListViews()` and `executeListView()` calls. It has the following properties:

Name	Type	Description
<code>ascendingLabel</code>	<code>string</code>	The localized type-specific label for sorting the column in ascending order. For example: "A-Z" for a text field, or "Low to High" for a numeric field. Set to null if the column isn't sortable.
<code>descendingLabel</code>	<code>string</code>	The localized type-specific label for sorting the column in ascending order. For example: "Z-A" for a text field, or "High to Low" for a numeric field. Set to null if the column is not sortable.
<code>fieldNameOrPath</code>	<code>string</code>	The field name or SOQL field path for the column.
<code>hidden</code>	<code>boolean</code>	If true, specifies that the column is not displayed, and is present only to support the display of other columns or other client-side logic.
<code>label</code>	<code>string</code>	The localized display label for the column.
<code>searchable</code>	<code>boolean</code>	Whether the column is searchable.
<code>selectListItem</code>	<code>string</code>	The SOQL SELECT item for the column. The item might differ from the field name or path, due to display formatting (for example, <code>toLabel</code> for picklists).
<code>sortDirection</code>	<code>orderByDirection</code>	An enumerated value, one of the following if the column is sortable: <ul style="list-style-type: none"> • <code>ascending</code> • <code>descending</code> Set to null if the column is not sortable.
<code>sortIndex</code>	<code>int</code>	The zero-based index that indicates the column's position within a multilevel sort, or null if the records are not sorted by the column.

Name	Type	Description
sortable	boolean	Whether the column is sortable, in which case it might be referenced in the <code>ExecuteListView orderBy</code> parameter.
type	FieldType	The column data type.

ListViewRecord

Represents a single row in a list view.

The `ListViewRecord` object is a member of the [ExecuteListViewResult](#) object and has the following properties:

Name	Type	Description
columns	ListViewRecordColumn[]	The columns and their values for the record. The record data columns are returned in the same order as metadata and describe columns. For any data column that's obtained by using <code>ExecuteListViewResult.getRecords()[0].getColumns[index]</code> , the corresponding describe column can be obtained with <code>ExecuteListViewResult.getColumns[index]</code> .

ListViewRecordColumn

Represents a single cell in a row from a list view.

The `ListViewRecordColumn` object is one cell (column) of a row ([ListViewRecord](#)) and has the following properties:

Name	Type	Description
fieldNameOrPath	string	The field name or SOQL field path for the column.
value	string	The contents of the record for a certain column, localized if appropriate, or null if there's no value.

findDuplicates()

Performs rule-based searches for duplicate records. The input is an array of [sObject](#), each of which specifies the values to search for and the type of object that supplies the duplicate rules. The output identifies the detected duplicates for each object that supplies the duplicate rules. `findDuplicates()` applies the rules to the values to do the search. The output identifies the detected duplicates for each `sObject`.

Syntax

```
FindDuplicatesResult[] duplicateResults =
    connection.findDuplicates(sObject[] inputSObjectArray);
```

Usage

Use `findDuplicates()` to apply duplicate rules associated with an object to values specified by each `sObject`. Each `sObject` also has a type that corresponds to an object.

`findDuplicates()` uses the duplicate rules for the object that has the same type as the `sObject`. For example, if the `sObject` type is `Account`, `findDuplicates()` uses the duplicate rules associated with the `Account` object.

Note:

- All the `sObject` elements in the input array must have the same type, and that type must correspond to an object type that supports duplicate rules.
- The input array is limited to 50 elements. If you exceed this limit, the SOAP call returns an [API Fault Element](#) containing the following fields:
 - `ExceptionCode`: `LIMIT_EXCEEDED`
 - `exceptionMessage`: `Configuration error: The number of records to check is greater than the permitted batch size.`

For each input `sObject`, `findDuplicates()` adds a `FindDuplicatesResult` object to the output array.

Matching is controlled by the values specified in the `sObject`. The values can include a record ID, a field map, or both. The specified values determine the behavior of `findDuplicates()`:

Record ID only

`findDuplicates()` searches the object defined by the duplicate rule for an existing record that has the same ID. Then it loads the values from that record, and searches for duplicates based on those values.

Field Map only

`findDuplicates()` loads the values from the map and searches for duplicates based on those values.

Record ID and Field Map

`findDuplicates()` searches the object defined by the duplicate rule for an existing record that has the same ID. It loads any values from that record that aren't specified in the map, and then loads values from the map. Based on the resulting union of values, `findDuplicates()` searches for duplicates.

The output of `findDuplicates()` is an array of `FindDuplicatesResult` objects with the same number of elements as the input array, and in the same order. The output objects encapsulate record IDs for duplicate records, if any. Optionally, the output objects also contain values from the duplicate records.

Each `FindDuplicatesResult` element contains a [DuplicateResult](#) object. If `findDuplicates()` doesn't find any duplicates for an `sObject`, the `duplicateRule` field in `DuplicateResult` contains the name of the duplicate rule that `findDuplicates()` applied, but the `matchResults` array is empty.

If the `includeRecordDetails` flag in [DuplicateRuleHeader](#) is set to `false`, `findDuplicates()` only returns the record IDs of the matching records. Otherwise, `findDuplicates()` returns all the fields specified in the primary [CompactLayout](#) associated with the target object.

Basic Steps for Using

1. Create one or more `sObject` objects with a type that corresponds to the object that has the duplicate rules you want to use.
2. In each `sObject`, specify record IDs or field maps (or both) to compare to records in the object.
3. Set [DuplicateRuleHeader](#) to control the output you want.

Sample

The following Java sample demonstrates how to search for duplicates of a Lead, using the standard Leads duplicate rule.

```

package wsc;

import com.sforce.soap.partner.*;
import com.sforce.soap.partner.Error;
import com.sforce.soap.partner.sobject.SObject;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;

public class Main {

    private static final String USERNAME = "YOUR-USERNAME";
    private static final String PASSWORD = "YOUR-PASSWORD&SECURITY-TOKEN";
    private static PartnerConnection connection = null;

    public static void main(String[] args) throws ConnectionException {

        // Create the configuration for the partner connection
        ConnectorConfig config = new ConnectorConfig();
        config.setUsername(USERNAME);
        config.setPassword(PASSWORD);

        // Initialize the connection
        connection = new PartnerConnection(config);

        SObject[] inputSObjectArray = new SObject[1];
        // Instantiate an empty Java SObject
        SObject searchCriteria = new SObject();
        // Set its type to Lead. This tells findDuplicates() to use the duplicate rules
        // for Lead
        searchCriteria.setType("Lead");
        /*
         * Set the necessary fields for matching, based on the standard matching rules for
         * Lead (Search
         * help.salesforce.com for "Standard Contact and Lead Matching Rule" to see the rules).
         */
        searchCriteria.setField("FirstName", "Marc");
        searchCriteria.setField("LastName", "Benioff");
        searchCriteria.setField("Company", "Salesforce.com Inc");
        searchCriteria.setField("Title", "CEO");
        searchCriteria.setField("Email", "ceo@salesforce.com");
        // Add the sObject to the input array
        inputSObjectArray[0] = searchCriteria;
        /*
         * By default, findDuplicates() returns only record IDs. To return additional values,
         * set the second parameter
         * to true.
         */
        connection.setDuplicateRuleHeader(
            /*
             * @param allowSave - Not Applicable for this API call

```


Arguments

Name	Type	Description
<code>sObjects</code>	Array of <code>sObject</code>	Required. A list of <code>sObject</code> objects that contain values you want to search for.

Response

An array of `FindDuplicatesResult` objects.

FindDuplicatesResult

Represents the result of a duplicate search for a single `sObject` in the input array. Because the object associated with the `sObject` can have more than one duplicate rule, `FindDuplicatesResult` contains an array of `DuplicateResult` objects.

Fields

Field Name	Field Type	Description
<code>duplicateResults</code>	Array of <code>DuplicateResult</code> objects	The result of each duplicate rule applied by <code>findDuplicates()</code> to a single <code>sObject</code> .
<code>errors</code>	Array of <code>Error</code> objects	Contains an array of errors encountered by <code>findDuplicates()</code> .
<code>success</code>	boolean	This field is set to <code>true</code> if the <code>findDuplicates()</code> doesn't encounter any errors.

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

[InvalidFieldFault](#)

findDuplicatesByIds()

Performs rule-based searches for duplicate records. The input is an array of IDs, each of which specifies the records for which to search for duplicates. The output identifies the detected duplicates for each object that supplies the duplicate rules.

`findDuplicatesByIds()` applies the rules to the record IDs to do the search. The output identifies the detected duplicates for each ID.

Syntax

```
FindDuplicatesResult[] duplicateResults =
    connection.findDuplicatesByIds(Id[] inputIdArray);
```

Usage

Use `findDuplicatesByIds()` to apply duplicate rules associated with an object to records represented by the record IDs.

`findDuplicatesByIds()` uses the duplicate rules for the object that has the same type as the input record IDs. For example, if the record ID represents an Account, `findDuplicatesByIds()` uses the duplicate rules associated with the Account object.

Note:

- All record IDs in the input array must have the same object type, and that type must correspond to an object type that supports duplicate rules.
- The input array is limited to 50 elements. If you exceed this limit, the SOAP call returns an [API Fault Element](#) containing the following fields:
 - `ExceptionCode: LIMIT_EXCEEDED`
 - `exceptionMessage: Configuration error: The number of records to check is greater than the permitted batch size.`

For each input ID, `findDuplicatesByIds()` adds an object to the output array.

Matching is controlled by the values specified by the input record ID. The values can include a record ID only.

`findDuplicatesByIds()` searches the object defined by the duplicate rule for an existing record that has the same ID. Then it loads the values from that record, and searches for duplicates based on those values.

The output of `findDuplicatesByIds()` is an array of objects with the same number of elements as the input array, and in the same order. The output objects encapsulate record IDs for duplicate records. Optionally, the output objects also contain values from the duplicate records.

Each element contains a [DuplicateResult](#) object. If `findDuplicatesByIds()` doesn't find any duplicates for an sObject, the `duplicateRule` field in `DuplicateResult` contains the name of the duplicate rule that `findDuplicatesByIds()` applied, but the `matchResults` array is empty.

If the `includeRecordDetails` flag in [DuplicateRuleHeader](#) is set to `false`, `findDuplicatesByIds()` returns only the record IDs of the matching records. Otherwise, `findDuplicatesByIds()` returns all the fields specified in the primary [CompactLayout](#) associated with the target object.

Basic Steps for Using

1. Create one or more ID objects that correspond to the object that has the duplicate rules you want to use.
2. Specify record IDs to compare to records in the object.
3. Set `DuplicateRuleHeader` to control the output you want.

Sample

The following Java sample demonstrates how to search for duplicates of a Lead, using the standard Leads duplicate rule.

```
package wsc;

import com.sforce.soap.partner.*;
import com.sforce.soap.partner.Error;
import com.sforce.soap.partner.sobject.SObject;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;

public class Main {

    private static final String USERNAME = "YOUR-USERNAME";
    private static final String PASSWORD = "YOUR-PASSWORD&SECURITY-TOKEN";
    private static PartnerConnection connection = null;

    public static void main(String[] args) throws ConnectionException {

        // Create the configuration for the partner connection
        ConnectorConfig config = new ConnectorConfig();
        config.setUsername(USERNAME);
        config.setPassword(PASSWORD);

        // Initialize the connection
        connection = new PartnerConnection(config);

        SObject[] objectsToSearch = new SObject[2];
        String[] inputIds = new String[2];
        // Instantiate an empty Java SObject
        SObject searchCriteria = new SObject();
        // Set its type to Lead. This tells findDuplicatesByIds() to use the duplicate rules
        // for Lead
        searchCriteria.setType("Lead");
        /*
         * Set the necessary fields for matching, based on the standard matching rules for
Lead
         * (Search help.salesforce.com for "Standard Contact and Lead Matching Rule" to see
the
         * rules).
         */
        searchCriteria.setField("FirstName", "Marc");
        searchCriteria.setField("LastName", "Benioff");
        searchCriteria.setField("Company", "Salesforce.com Inc");
        searchCriteria.setField("Title", "CEO");
        searchCriteria.setField("Email", "ceo@salesforce.com");
        // Add the sObjects to the input array
        objectsToSearch[0] = searchCriteria;
        objectsToSearch[1] = searchCriteria;

        SaveResult[] saveResults = connection.create(objectsToSearch);

        for (int i = 0; i < saveResults.length; ++i) {
            if (saveResults[i].isSuccess()) {
```

```

        System.out.println("Successfully created ID: " + saveResults[i].getId());
        inputIds[i] = saveResults[i].getId();
    } else {
        System.out.println("Error: could not create SObject.");
        System.out.println("The error reported was: " +
            saveResults[i].getErrors()[0].getMessage() + "\n");
    }
}
/*
 * By default, findDuplicatesByIds() returns only record IDs. To return additional
values,
 * set the second parameter to true.
 */
connection.setDuplicateRuleHeader(
    /*
     * @param allowSave - Not Applicable for this API call
     */
    false,
    /* @param includeRecordDetails */
    false,
    /*
     * @param runAsCurrentUser - Not Applicable for this API call
     */
    false);

// Invoke findDuplicatesByIds() to find duplicates based on the information in the
// SObject array
FindDuplicatesResult[] callResults = connection.findDuplicatesByIds(inputIds);

// Iterate through the results
/* For each Id in the input array, get the duplicate results. There could be more
matches
 * depending on the data in the organization.
 */
for (FindDuplicatesResult findDupeResult : callResults) {
    // If errors were found for this Id, print them out
    if (!findDupeResult.isSuccess()) {
        for (Error findDupError : findDupeResult.getErrors()) {
            System.out.println("FindDuplicatesRule errors detected: " +
findDupError.getMessage());
        }
    } else {
        /*
         * Get the DuplicateResult object array for the result. Each element in the array
represents
         * the result of testing one duplicate rule for the Id. Process each DuplicateResult.
         */
        for (DuplicateResult dupeResult : findDupeResult.getDuplicateResults()) {
            System.out.println("Duplicate rule: " + dupeResult.getDuplicateRule());
            // Print out the name of the object associated with the duplicate
            // rule
            System.out.println("Source of this duplicate rule is: " +
                dupeResult.getDuplicateRuleEntityType());
        }
    }
}

```



```

    for (MatchResult matchResult : dupeResult.getMatchResults()) {
        if (!matchResult.isSuccess()) {
            for (Error e : matchResult.getErrors()) {
                System.out.println("Errors detected: " + e.getMessage());
            }
        } else {
            System.out.println("Matching rule is: " + matchResult.getRule());
            System.out.println("Object type for this matching rule is: " +
matchResult.getEntityType());
            for (MatchRecord matchRecord : matchResult.getMatchRecords()) {
                System.out.println("Duplicate record ID: " +
matchRecord.getRecord().getId());
            }
        }
    }
}
}
}
}
}
}
}
}

```

Arguments

Name	Type	Description
IDs	Array of ID	Required. A list of IDs that contain values you want to search for.

Response

An array of `FindDuplicatesResult` objects.

FindDuplicatesResult

Represents the result of a duplicate search for a single ID in the input array. Because the object associated with the sObject can have more than one duplicate rule, `FindDuplicatesResult` contains an array of [DuplicateResult](#) objects.

Fields

Field Name	Field Type	Description
<code>duplicateResults</code>	Array of DuplicateResult objects	The result of each duplicate rule applied by <code>findDuplicatesByIds()</code> to a single sObject.
<code>errors</code>	Array of Error objects	Contains an array of errors encountered by <code>findDuplicatesByIds()</code> .

Field Name	Field Type	Description
success	boolean	This field is set to <code>true</code> if <code>findDuplicatesByIds()</code> doesn't encounter any errors.

Faults

[InvalidObjectFault](#)

[UnexpectedErrorFault](#)

[InvalidFieldFault](#)

getDeleted()

Retrieves the list of individual records that have been deleted within the given timespan for the specified object.

Syntax

```
GetDeletedResult = connection.getDeleted(string sObjectType, dateTime startDate, dateTime
  endDate);
```

Usage

Use `getDeleted()` for data replication applications to retrieve a list of records that have been deleted from your organization's data within the specified timespan. The `getDeleted()` call retrieves a `GetDeletedResult` object that contains an array of `DeletedRecord` objects containing the ID of each deleted record and the date/time (Coordinated Universal Time (UTC) time zone) on which it was deleted. Be sure to read [Data Replication](#) before using `getDeleted()` in your client applications. (For information on IDs, see [ID Field Type](#).)

As of release 8.0, the `getDeleted()` call respects the user's sharing model.

Rules and Guidelines

When replicating deleted records, consider the following rules and guidelines:

- The specified `startDate` must chronologically precede the specified `endDate` value by more than one minute. The specified `startDate` can't be the same value as, or later than, the specified `endDate` value. Otherwise, the API returns an `INVALID_REPLICATION_DATE` error.
- Records are returned only if the user has access to them.
- 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). If the purge has been performed before your `getDeleted()` call is executed, an `INVALID_REPLICATION_DATE` error is returned.
- If `latestDateCovered` is less than `endDate`, the call fails, returning an `INVALID_REPLICATION_DATE` error with the value of `latestDateCovered`.
- Deleted records are written to a delete log, which `getDeleted()` accesses. A background process that runs every two hours purges records that have been in an organization's delete log for more than two hours if the number of records is above a certain

limit. Starting with the oldest records, the process purges delete log entries until the delete log is back below the limit. This is done to protect Salesforce from performance issues related to massive delete logs. The limit is calculated using this formula:

$$5000 * \text{number of licenses in the organization}$$

For example, an organization with 1,000 licenses could have up to 5,000,000 (five million) records in the delete log before any purging took place. If purging has been performed before your `getDeleted()` call is executed, an `INVALID_REPLICATION_DATE` error is returned. If you get this exception, you should do a full pull of the table.

- If you delete a large number of records, your data replication should run more frequently than every two hours to ensure all records are returned by `getDeleted()`.
- Client applications typically poll for changed data periodically. For important polling considerations, see [Polling for Changes](#).
- Records for certain objects can't be replicated via the API. To replicate a record via the `getDeleted()` call, its object must be configured as replicatable (`replicatable` is `true`). To determine whether a given object can be replicated, your client application can invoke the `describeObjects()` call on the object and inspect its `replicatable` property.
- Development tools differ in the way that they handle time data. Some development tools report the local time, while others report only the Coordinated Universal Time (UTC) time. To determine how your development tool handles time values, refer to its documentation.
- If you call `getDeleted()` for a history object, the call returns the records deleted during the given date range for all history objects, not only the history object you specified. For example, if you call `getDeleted()` for `AccountHistory`, you'll get records deleted during the given date range for `AccountHistory`, `ContactHistory`, and so on. However, `getDeleted()` calls on `OpportunityHistory` return only deleted `OpportunityHistory` records, not other associated deleted history objects.

Basic Steps for Replicating Deleted Records

You can replicate deleted records using the following basic steps for each object:

1. Optionally, determine whether the structure of the object has changed since the last replication request, as described in [Checking for Structural Changes in the Object](#).
2. Call `getDeleted()`, passing in the object and the relevant time span for deleted records.
3. In the `DeleteResult` object, iterate through the returned array of `DeletedRecord` objects containing the ID of each deleted record and the date on which it was deleted (Coordinated Universal Time (UTC) time zone).
4. Take the appropriate action on the local data to remove the deleted records or flag as deleted.
5. Optionally, save the request time span for future reference, using the value of `latestDateCovered`.

A client application likely performs other tasks associated with data replication operations. For example, if an opportunity is closed, a client application might run a new revenue report. Similarly, if a task is completed, the process might log this in another system.

Sample Code—Java

This sample calls `getDeleted()` to get all accounts that were deleted in the last 60 minutes. It then writes the ID and the deleted date of each returned account to the console.

```
public void getDeletedRecords() {
    try {
        GregorianCalendar endTime = (GregorianCalendar)
            connection.getServerTimestamp().getTimestamp();
        GregorianCalendar startTime = (GregorianCalendar) endTime.clone();
        // Subtract 60 minutes from the server time so that we have
        // a valid time frame.
```

```

startTime.add(GregorianCalendar.MINUTE, -60);
System.out.println("Checking deletes at or after: "
    + startTime.getTime().toString());

// Get records deleted during the specified time frame.
GetDeletedResult gdResult = connection.getDeleted("Account",
    startTime, endTime);

// Check the number of records contained in the results,
// to check if something was deleted in the 60 minute span.
DeletedRecord[] deletedRecords = gdResult.getDeletedRecords();
if (deletedRecords != null && deletedRecords.length > 0) {
    for (int i = 0; i < deletedRecords.length; i++) {
        DeletedRecord dr = deletedRecords[i];
        System.out.println(dr.getId() + " was deleted on "
            + dr.getDeletedDate().getTime().toString());
    }
} else {
    System.out.println("No deletions of Account records in "
        + "the last 60 minutes.");
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

Sample Code—C#

This sample calls `getDeleted()` to get all accounts that were deleted in the last 60 minutes. It then writes the ID and the deleted date of each returned account to the console.

```

public void getDeletedRecords()
{
    try
    {
        DateTime endTime = binding.getServerTimestamp().timestamp;
        // Subtract 60 minutes from the server time so that we have
        // a valid time frame.
        DateTime startTime = endTime.AddMinutes(-60);
        Console.WriteLine("Checking deletes at or after: "
            + startTime.ToLocalTime().ToString());

        // Get records deleted during the specified time frame.
        GetDeletedResult gdResult = binding.getDeleted("Account",
            startTime, endTime);

        // Check the number of records contained in the results,
        // to check if something was deleted in the 60 minute span.
        DeletedRecord[] deletedRecords = gdResult.deletedRecords;
        if (deletedRecords != null && deletedRecords.Length > 0)
        {
            for (int i = 0; i < deletedRecords.Length; i++)
            {

```

```

        DeletedRecord dr = deletedRecords[i];
        Console.WriteLine(dr.id + " was deleted on "
            + dr.deletedDate.ToLocalTime().ToString());
    }
}
else
{
    Console.WriteLine("No deletions of Account records in "
        + "the last 60 minutes.");
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

Arguments

Name	Type	Description
sObjectTypeEntityType	string	Object type. The specified value must be a valid object for your organization. See sObject .
startDate	dateTime	Starting date/time (Coordinated Universal Time (UTC)—not local— timezone) of the timespan for which to retrieve the data. The API ignores the seconds portion of the specified dateTime value (for example, 12:30:15 is interpreted as 12:30:00 UTC).
endDate	dateTime	Ending date/time (Coordinated Universal Time (UTC)—not local— timezone) of the timespan for which to retrieve the data. The API ignores the seconds portion of the specified dateTime value (for example, 12:35:15 is interpreted as 12:35:00 UTC).

Limits

When a `getDeleted()` call returns too many results, the exception `EXCEEDED_ID_LIMIT` is returned in the response. See [API Call Limits](#) for the number of records that can be returned.

Response

[GetDeletedResult](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[Data Replication](#)

[API Call Basics](#)

GetDeletedResult

The `getDeleted()` call returns a `GetDeletedResult` object that contains an array of `DeletedRecord` records and two properties:

Name	Type	Description
<code>earliestDateAvailable</code>	<code>dateTime</code>	For the object type of the <code>getDeleted()</code> call, the timestamp (Coordinated Universal Time (UTC)—not local—timezone) of the last physically deleted object. If this value is less than <code>endDate</code> , the call will fail, and you should resynch your data before performing another replication.
<code>deletedRecords[]</code>	<code>deletedRecords=</code>	Array of the deleted records which satisfy the start and end dates specified in the <code>getDeleted()</code> call.
<code>latestDateCovered</code>	<code>dateTime</code>	The timestamp (Coordinated Universal Time (UTC)—not local—time zone) of the last date covered in the <code>getDeleted()</code> call. If there is a value, it is less than or equal to <code>endDate</code> . A value here indicates that, for safety, you should use this value for the <code>startDate</code> of your next call to capture the changes that started after this date but did not complete before <code>endDate</code> and were, therefore, not returned in the previous call.

deletedRecords

The `GetDeletedResult` contains an array of `deletedRecords`, which contain the following properties:

Name	Type	Description
<code>deletedDate</code>	<code>dateTime</code>	Date and time (Coordinated Universal Time (UTC)—not local—timezone) when this record was deleted. This information is obtained using the <code>SystemModstamp</code> system field if available.
<code>id</code>	ID	ID of an <code>sObject</code> that has been deleted.

getUpdated()

Retrieves the list of individual records that have been updated (added or changed) within the given timespan for the specified object.

Syntax

```
GetUpdatedResult[] = connection.getUpdated(string sObjectType, dateTime startDate, dateTime
  endDate);
```

Usage

Use `getUpdated()` for data replication applications to retrieve a set of IDs for objects of the specified object that have been created or updated within the specified timespan. The `getUpdated()` call retrieves an array of `GetUpdatedResult` objects containing the ID of each created or updated object and the date/time (Coordinated Universal Time (UTC) time zone) on which it was created or updated, respectively. Be sure to read [Data Replication](#) before using `getUpdated()` in your client application.



Note: The `getUpdated()` call retrieves the IDs only for objects to which the logged-in user has access.

Rules and Guidelines

When replicating created and updated objects, consider the following rules and guidelines:

- The specified `startDate` must chronologically precede the specified `endDate` value. The specified `startDate` cannot be the same value as, or later than, the specified `endDate` value. Otherwise, the API returns an `INVALID_REPLICATION_DATE` error.
- Results are returned for no more than 30 days previous to the day the call is executed.
- Client applications typically poll for changed data periodically. For important polling considerations, see [Polling for Changes](#).
- Your client application can replicate any objects to which it has sufficient permissions. For example, to replicate all data for your organization, your client application must be logged in with “View All Data” access rights to the specified object. Similarly, the objects must be within your sharing rules. For more information, see [Factors that Affect Data Access](#).
- Certain objects cannot be replicated via the API. To replicate an object via the `getUpdated()` call, its object must be configured as replicatable (`replicatable` is `true`). To determine whether a given object can be replicated, your client application can invoke the `describeObjects()` call on the object and inspect its `replicatable` property.
- Certain objects cannot be deleted, such as [Group](#), [User](#), [Contract](#), or [Product2](#) objects. However, if instances of these objects are no longer visible in the Salesforce user interface, they may have been rendered inactive so that only users with administrative access can see them. To determine whether a missing object instance has been made inactive, your client application can call `getUpdated()` and check the object’s active flag.
- Development tools differ in the way that they handle time data. Some development tools report the local time, while others report only the Coordinated Universal Time (UTC) time. To determine how your development tool handles time values, refer to its documentation.

Basic Steps for Replicating Updated Objects

Replicating objects involves the following basic steps for each object that you want to replicate:

1. Optionally, the client application determines whether the structure of the object has changed since the last replication request, as described in [Checking for Structural Changes in the Object](#).
2. Call `getUpdated()`, passing in the object and timespan for which to retrieve data.
3. Iterate through the returned array of IDs. For each ID element in the array, call `retrieve()` to obtain the latest information you want from the associated object. Your client application must then take the appropriate action on the local data, such as inserting new rows or updating existing ones with the latest information.

- Optionally, the client application saves the request timestamp for future reference.

A client application likely performs other tasks associated with data replication operations. For example, if an opportunity were to become closed, a client application might run a new revenue report. Similarly, if a task were completed, the process might log this somehow in another system.

Sample Code—Java

This sample gets the accounts that were updated in the last 60 minutes and writes their IDs to the console.

```
public void getUpdatedRecords() {
    try {
        GregorianCalendar endTime = (GregorianCalendar) connection
            .getServerTimestamp().getTimestamp();
        GregorianCalendar startTime = (GregorianCalendar) endTime.clone();
        // Subtract 60 minutes from the server time so that we have
        // a valid time frame.
        startTime.add(GregorianCalendar.MINUTE, -60);
        System.out.println("Checking updates as of: "
            + startTime.getTime().toString());

        // Get the updated accounts within the specified time frame
        GetUpdatedResult ur = connection.getUpdated("Account", startTime,
            endTime);
        System.out.println("GetUpdateResult: " + ur.getIds().length);

        // Write the results
        if (ur.getIds() != null && ur.getIds().length > 0) {
            for (int i = 0; i < ur.getIds().length; i++) {
                System.out.println(ur.getIds()[i] + " was updated between "
                    + startTime.getTime().toString() + " and "
                    + endTime.getTime().toString());
            }
        } else {
            System.out.println("No updates to accounts in "
                + "the last 60 minutes.");
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample gets the accounts that were updated in the last 60 minutes and writes their IDs to the console.

```
public void getUpdatedRecords()
{
    try
    {
        DateTime endTime = binding.getServerTimestamp().timestamp;
        // Subtract 60 minutes from the server time so that we have
        // a valid time frame.
```



```

DateTime startTime = endTime.AddMinutes(-60);
Console.WriteLine("Checking updates as of: "
    + startTime.ToLocalTime().ToString());

// Get the updated accounts within the specified time frame
GetUpdatedResult ur = binding.getUpdated("Account", startTime,
    endTime);
Console.WriteLine("GetUpdateResult: " + ur.ids.Length);

// Write the results
if (ur.ids != null && ur.ids.Length > 0)
{
    for (int i = 0; i < ur.ids.Length; i++)
    {
        Console.WriteLine(ur.ids[i] + " was updated between "
            + startTime.ToLocalTime().ToString() + " and "
            + endTime.ToLocalTime().ToString());
    }
}
else
{
    Console.WriteLine("No updates to accounts in "
        + "the last 60 minutes.");
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

Arguments

Name	Type	Description
sObjectTypeEntityType	string	Object type. The specified value must be a valid object for your organization. For a list of standard objects, see Standard Objects .
startDate	dateTime	Starting date/time (Coordinated Universal Time (UTC) time zone—not local—timezone) of the timespan for which to retrieve the data. The API ignores the seconds portion of the specified dateTime value (for example, 12:30:15 is interpreted as 12:30:00 UTC).
endDate	dateTime	Ending date/time (Coordinated Universal Time (UTC) time zone—not local—timezone) of the timespan for which to retrieve the data. The API ignores the seconds portion of the specified dateTime value (for example, 12:35:15 is interpreted as 12:35:00 UTC).

! **Important:** There is a limit of 600,000 IDs in the result [GetUpdatedResult\[\]](#). If your `getUpdated()` call returns more than 600,000 IDs, an exception `EXCEEDED_ID_LIMIT` is returned. You can correct the error by choosing start and end dates that are closer together.

Response

[GetUpdatedResult\[\]](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)


SEE ALSO:

[Data Replication](#)

[API Call Basics](#)

GetUpdatedResult

The `getUpdated()` call returns a `GetUpdatedResult` object that contains information about each record that was inserted or updated within the given timespan. An `GetUpdatedResult` object has the following properties:

Name	Type	Description
<code>id[]</code>	ID	Array of IDs of each object that has been updated.
<code>latestDateCovered</code>	<code>dateTime</code>	The timestamp (Coordinated Universal Time (UTC)—not local— time zone) of the last date covered in the <code>getUpdated()</code> call. If there is a value, it is less than or equal to <code>endDate</code> . A value here indicates that, for safety, you should use this value for the <code>startDate</code> of your next call to capture the changes that started after this date but did not complete before the <code>endDate</code> and were, therefore, not returned in the previous call.  Note: If Salesforce executes a long-running transaction on your instance, the value in this field is the start time of that long-running transaction until it completes. This is because a long-running transaction might affect your user data (for example, batch processing).

`invalidateSessions()`

Ends one or more sessions specified by a `sessionId`.

Syntax

```
InvalidateSessionsResult = connection.invalidateSessions(string[] sessionId);
```

Usage

Use this call to end one or more sessions.

You can also use `logout ()` to end just one session, the session of the logged-in user.

Sample Code—Java

This sample invalidates a set of sessions. The method in this sample takes an array of session IDs passed in as String values. The method then calls `invalidateSessions ()` with this array and then checks the results for any errors.

```
public void invalidateSessionsSample(String[] sessionIds) {
    try {
        InvalidateSessionsResult[] results;
        results = connection.invalidateSessions(sessionIds);
        for (InvalidateSessionsResult result : results) {
            // Check results for errors
            if (!result.isSuccess()) {
                if (result.getErrors().length > 0) {
                    System.out.println("Status code: "
                        + result.getErrors()[0].getStatusCode());
                    System.out.println("Error message: "
                        + result.getErrors()[0].getMessage());
                }
            } else {
                System.out.println("Success.");
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample invalidates a set of sessions. The method in this sample takes an array of session IDs passed in as String values. The method then calls `invalidateSessions ()` with this array and then checks the results for any errors.

```
public void invalidateSessionsSample(string[] sessionIds)
{
    try
    {
        InvalidateSessionsResult[] results;
        results = binding.invalidateSessions(sessionIds);
        foreach (InvalidateSessionsResult result in results)
        {
            // Check results for errors
            if (!result.success)
            {
                if (result.errors.Length > 0)
                {
                    Console.WriteLine("Status code: " +
                        result.errors[0].statusCode);
                    Console.WriteLine("Error message: " +
                        result.errors[0].message);
                }
            }
        }
    }
}
```

```

        else
        {
            Console.WriteLine("Success.");
        }
    }
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

Arguments

Name	Type	Description
sessionIds	string[]	One or more sessionId strings. Limit 200. You can obtain your <code>sessionId</code> from the SessionHeader .

Response

[InvalidateSessionsResult\[\]](#)

Faults

[UnexpectedErrorFault](#)

InvalidateSessionsResult

The `invalidateSessions()` call returns an array of `LogoutResult` objects. Each object has the following properties:

Name	Type	Description
success	boolean	Indicates whether the session was successfully terminated (<code>true</code>) or not (<code>false</code>).
errors	Error[]	If an error occurred during the call, an array of one or more Error objects. Each object contains an error code and description.

login ()

Logs in to the login server and starts a client session.

 **Note:** `login ()` calls count toward your login rate limit.

Syntax

```
LoginResult = connection.login(string username, string password);
```

Usage

Use the `login()` call to log in to the login server and start a client session. The client app logs in and obtains a `sessionId` and server URL before making other API calls.


When a client app invokes the `login()` call, it passes in a username and password as credentials. Upon invocation, the API authenticates the credentials. It then returns the `sessionId`, the user ID associated with the logged-in username, and a URL that points to the Lightning Platform API to use in all subsequent API calls.

Salesforce checks the IP address from which the client app is logging in and blocks logins from unknown IP addresses. If the API blocks the login, Salesforce returns a login fault. To log in, the user must add the security token at the end of the user's password. For example, if a user's password is `mypassword` and the security token is `XXXXXXXXXX`, the user enters `mypasswordXXXXXXXXXX`. Users get their security token by changing their password or resetting their security token from the Salesforce user interface. When users change their password or reset their security token, Salesforce sends a new security token to the email address on the user's Salesforce record. The security token is valid until the user resets the security token, or changes the password, or you reset the user's password. When the security token is invalid, the user must repeat the login process. To avoid another log in, add the client's IP address to the org's list of trusted IP addresses. For more information, see [Security Token](#).

After logging in, make sure that your client app performs these tasks.

- Sets the session ID in the SOAP header so that the API can validate subsequent requests for this session.
- Specifies the server URL as the target for subsequent service requests. The login server supports only login calls.

Development tools differ in the way you specify session headers and server URLs. For more information, see the documentation for your particular development tool.

 **Note:** Multiple client apps can log in using the same `username` argument. However, this approach increases your risk of getting errors due to query limits. A user can have up to 10 query cursors open at a time. If 10 `QueryLocator` cursors are open when a client application, logged in as the same user, attempts to open a new one, then the oldest of the 10 cursors is released. If the client application attempts to open the released query cursor, an error results.

The limit is 3,600 calls to `login()` per user per hour. Exceeding this limit results in a "Login Rate Exceeded" error. After reaching the hourly limit, Salesforce blocks the user from logging in. Users can try to log in again an hour after the block occurred.

Enterprise and Partner Endpoints


In API version 11.1 and earlier, client apps built with the partner WSDL can send requests to the enterprise endpoint, and enterprise WSDL apps can send requests to the partner endpoint. Beginning with version 12.0, this functionality is not supported.

Endpoint Base URLs

When specifying an endpoint for a Salesforce org, there are three options for the base URL.

1. Recommended: Your My Domain login URL, in the format `https://MyDomainName.my.salesforce.com` for production orgs and `https://MyDomainName--SandboxName.sandbox.my.salesforce.com` for sandboxes with enhanced domains enabled. If you're not using enhanced domains, your sandbox, use `https://MyDomainName--SandboxName.my.salesforce.com`, and plan to update these endpoints when you enable enhanced domains.

- The default Salesforce login URLs: `https://login.salesforce.com` for production and Developer Edition orgs and `https://test.salesforce.com` for sandboxes.

 **Note:** Admins can prevent SOAP API logins from the default Salesforce login URLs via a My Domain setting.

All examples use the recommended My Domain login URL format for a production org. To specify an endpoint for a sandbox or to use the default Salesforce login URLs, modify the example as needed.

To understand the benefits of using your My Domain login URL versus the default Salesforce login URL, see [Log In to Salesforce with Code](#) in Salesforce Help.

Login When Using a Proxy


If you log in to Salesforce via a proxy, set the proxy host and port number on the instance of the `ConnectorConfig` class that you use to log in. If you must authenticate on the proxy, set the username and password.

```
ConnectorConfig config = new ConnectorConfig();
config.setUsername(userId);
config.setPassword(pwd);
config.setAuthEndpoint(authEndPoint);
config.setProxy(proxyHost, proxyPort);
// Set the username and password if your proxy must be authenticated
config.setProxyUsername(proxyUsername);
config.setProxyPassword(proxyPassword);
try {
    EnterpriseConnection connection = new EnterpriseConnection(config);
    // etc.
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
```

Session Expiration

Client apps aren't required to explicitly log out to end a session. Sessions expire automatically after a predetermined length of inactivity. The default is two hours. If you make an API call, the inactivity timer is reset to zero. To change the session expiration (timeout) value, from Setup, enter *Session Settings* in the Quick Find box, and select **Session Settings**.

Active Self-Service Users Authentication

 **Note:** Starting with Spring '12, the Self-Service portal isn't available for new Salesforce orgs. Existing orgs continue to have access to the Self-Service portal.

To authenticate active self-service users, use `LoginScopeHeader` to specify the `Organization` ID against which self-service users are authenticated. A self-service user must exist and be active before being authenticated (see [SelfServiceUser](#)).

Customer Experience Cloud Site User Authentication

To authenticate an active Experience Cloud site user who has the API Enabled permission, use `LoginScopeHeader` to specify the `Organization` ID of the org with Experience Cloud sites. Site users must exist, be active, and belong to the Experience Cloud site before being authenticated.

When specifying an endpoint that authenticates an Experience Cloud site user, the base URL is different in orgs without enhanced domains enabled.

- If enhanced domains are enabled, the base URL format is `https://MyDomainName.my.site.com` for a production org and `https://MyDomainName--SandboxName.sandbox.my.site.com` for a sandbox org.
- If enhanced domains aren't enabled, the base URL format is `https://ExperienceCloudSitesSubdomain.force.com` for a production org and `SandboxName-CommunitySubdomainName.InstanceName.force.com` for a sandbox org.

All examples for Experience Cloud Site User Authorization use the base URL format for a production org with enhanced domains enabled. To specify an endpoint for a sandbox org, or if you haven't yet enabled enhanced domains, update the base URL.

Example Endpoints

Client apps can send login requests to these endpoints (using valid values for the authentication endpoint).

Enterprise WSDL:

- `String authEndPoint = "https://MyDomainName.my.salesforce.com/services/Soap/c/version/"`
- `String authEndPoint = "https://MyDomainName.my.site.com/path-prefix/services/Soap/c/version/"`

Partner WSDL:

- `String authEndPoint = "https://MyDomainName.my.salesforce.com/services/Soap/u/version/"`
- `String authEndPoint = "https://MyDomainName.my.site.com/path-prefix/services/Soap/u/version/"`

Logout

Salesforce recommends that you always call `logout()` to end a session when it's no longer needed. This call ends any child sessions. To provide the most protection, log out the user instead of waiting for the session to expire.

Sample Code—Java

This sample logs a user in with the specified username, password, and authentication endpoint URL. The sample writes user and session information to the console after a successful login. Before running this sample, replace the values for username, password, and authentication endpoint with valid values.

To learn how to generate and import the web service WSDL needed to make API calls, see [Step 2: Generate or Obtain the Web Service WSDL](#) in the Quick Start.

```
public boolean loginSample() {
    boolean success = false;
    String username = "username";
    String password = "password";
    String authEndPoint = "https://MyDomainName.my.salesforce.com/services/Soap/c/24.0/";

    try {
        ConnectorConfig config = new ConnectorConfig();
        config.setUsername(username);
    }
}
```

```

config.setPassword(password);

System.out.println("AuthEndPoint: " + authEndPoint);
config.setAuthEndpoint(authEndPoint);

connection = new EnterpriseConnection(config);

// Print user and session info
GetUserInfoResult userInfo = connection.getUserInfo();
System.out.println("UserID: " + userInfo.getUserId());
System.out.println("User Full Name: " + userInfo.getUserFullName());
System.out.println("User Email: " + userInfo.getUserEmail());
System.out.println();
System.out.println("SessionID: " + config.getSessionId());
System.out.println("Auth End Point: " + config.getAuthEndpoint());
System.out
    .println("Service End Point: " + config.getServiceEndpoint());
System.out.println();

success = true;
} catch (ConnectionException ce) {
    ce.printStackTrace();
}

return success;
}

```

Sample Code—C#

This sample logs a user in using the specified username and password. The result of the login call contains the service endpoint URL, which is the virtual server instance that is servicing your org, and a unique session ID. The sample sets these returned values on the binding. It sets the binding URL to the returned service endpoint. It also sets the session ID on the session header that is used on all API calls. Next, the sample writes user and session information to the console after a successful login. Before running this sample, replace the values for username and password with valid values.

To learn how to generate and import the web service WSDL needed to make API calls, see [Step 2: Generate or Obtain the Web Service WSDL](#) in the Quick Start.

```

public bool loginSample()
{
    Boolean success = false;
    string username = "username";
    string password = "password";

    // Create a service object
    binding = new SforceService();

    LoginResult lr;
    try
    {
        Console.WriteLine("\nLogging in...\n");
        lr = binding.login(username, password);
    }
}

```



```

/**
 * The login results contain the endpoint of the virtual server instance
 * that is servicing your org. Set the URL of the binding
 * to this endpoint.
 */
// Save old authentication end point URL
String authEndPoint = binding.Url;
// Set returned service endpoint URL
binding.Url = lr.serverUrl;

/** Get the session ID from the login result and set it for the
 * session header that will be used for all subsequent calls.
 */
binding.SessionHeaderValue = new SessionHeader();
binding.SessionHeaderValue.sessionId = lr.sessionId;

// Print user and session info
GetUserInfoResult userInfo = lr.userInfo;
Console.WriteLine("UserID: " + userInfo.userId);
Console.WriteLine("User Full Name: " +
    userInfo.userFullName);
Console.WriteLine("User Email: " +
    userInfo.userEmail);
Console.WriteLine();
Console.WriteLine("SessionID: " +
    lr.sessionId);
Console.WriteLine("Auth End Point: " +
    authEndPoint);
Console.WriteLine("Service End Point: " +
    lr.serverUrl);
Console.WriteLine();

// Return true to indicate that we are logged in, pointed
// at the right URL and have our security token in place.
success = true;
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
return success;
}

```

Arguments

Name	Type	Description
username	string	Login username.
password	string	Login password associated with the specified username.

The login request size is limited to 10 KB.

Response

[LoginResult](#)

Faults

[LoginFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[API Call Basics](#)

LoginResult

The `login()` call returns a `LoginResult` object, which has the following properties:

Name	Type	Description
<code>metadataServerUrl</code>	string	URL of the endpoint that will process subsequent metadata API calls. Your client application needs to set the endpoint.
<code>passwordExpired</code>	boolean	Indicates whether the password used during the login attempt is expired (<code>true</code>) or not (<code>false</code>). If the password has expired, then the API returns a valid <code>sessionId</code> , but the only allowable operation is the <code>setPassword()</code> call.
<code>serverUrl</code>	string	URL of the endpoint that will process subsequent API calls. Your client application needs to set the endpoint.
<code>sessionId</code>	string	Unique ID associated with this session. Your client application needs to set this value in the session header.
<code>userId</code>	ID	ID of the user associated with the specified username and password.
<code>userInfo</code>	<code>getUserInfoResult</code>	User information fields. For a list of these fields, see getUserInfoResult .

logout()

Ends the session of the logged-in user.

Syntax

```
connection.logout();
```

Usage

This call ends the session for the logged-in user issuing the call. No arguments are needed.

To end one or more sessions started by someone other than the logged-in user, see [invalidateSessions\(\)](#).

Sample Code—Java

This sample calls `logout()` to log the current user out and writes a message to the console.

```
public void logoutSample() {
    try {
        connection.logout();
        System.out.println("Logged out.");
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample calls `logout()` to log the current user out and writes a message to the console.

```
public void logoutSample()
{
    try
    {
        binding.logout();
        Console.WriteLine("Logged out.");
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

Arguments

This call uses no arguments. It ends the session for the logged-in user issuing the call, so no arguments are needed. The logged-in user is identified by the `sessionId` specified in the [SessionHeader](#) for this call.

Response

Void is returned. Because failure of the call means that the session has already been logged out, no results are needed. Any unexpected error, such as system unavailability, throws an error that should be handled by your client application.

Faults

[UnexpectedErrorFault](#)

merge ()

! **Important:** Where possible, we changed noninclusive terms to align with our company value of Equality. Because changing terms in our code can break current implementations, we maintained this object's name.

Combines up to 3 records of the same type into 1 record. The input is an array of `MergeRequest` elements, each of which specifies the records to combine. The output is a `MergeResult` object that contains information about the result of the merge.

Syntax

```
MergeResult [] = connection.merge (MergeRequest [] mergeRequests);
```

Usage

Use `merge ()` to combine records of the same object type into one of the records, known as the *main record*. `merge ()` deletes the other records, known as the *victim records*. If a victim record has related records, `merge ()` makes the main record the new parent of the related records.

Rules and Guidelines

Values from non-main records

Before you call `merge ()`, decide if you want field values in the non-main records to supersede the main record values. If you do, set the field names and values in the record identified by the `masterRecord` of the `MergeRequest`.

Contacts, Leads, and Data Privacy Records

When you merge contacts or leads that have corresponding data privacy records based on the Individual object, `merge ()` determines the correct data privacy record to associate with the main record.

- If you selected the option to retain the most recently updated data privacy record for merging leads and contacts, `merge ()` selects the most recently updated data privacy record.
- Otherwise, `merge ()` selects the data privacy record already associated with the main record.

Successive merges

Because `merge ()` handles each `MergeResult` element in the input argument as a separate transaction, you can successively merge several records into the same main record.

To perform successive merges, call `merge ()` with an array of `MergeResult` elements. For each `MergeResult` element, set:

- `masterRecord` to the main record ID.
- Each element in the `recordToMergeIds` array to the ID of a record you want to combine into the main record.

Deleted records

Use `queryAll ()` to view records that have been deleted during a merge.

List merged records

To find all records that have been merged since a given point in time, call `queryAll ()` with a SELECT statement. For example:

```
SELECT Id FROM Contact WHERE isDeleted=true and masterRecordId != null
AND SystemModstamp > 2006-01-01T23:01:01+01:00
```

Supported Object Types

The supported object types are [Lead](#), [Contact](#), [Account](#), [Person Account](#), and [Individual](#). You can only merge objects of the same type. For example, leads can be merged only with leads.

Account Hierarchies

When you merge accounts that are part of an account hierarchy, `merge()` tries to set the victims' child records as children of the main record. If this action causes a cyclical relationship, `merge()` returns an error.

Contacts Reports To relationships

When you merge contacts that have a value for the `ReportsToId` field, `merge()` tries to add the victims' `ReportsToId` value to the main record. If this action causes a cyclical relationship, `merge()` reports an error.

Contacts and portal users

When you want to merge a contact victim record that has an associated portal user, set `AdditionalInformationMap` for victim record's `MergeRequest` element. You can only merge 1 victim with a portal user into the main record. In Salesforce Classic, you can't merge person accounts that are enabled to use a Customer Portal.


Considerations

The following limits apply to any merge request:

- Up to 200 merge requests can be made in a single SOAP call.
- Up to three records can be merged in a single request, including the main record. This limit is the same as the limit enforced by the Salesforce user interface. To merge more than 3 records, do a successive merge.
- External ID fields cannot be used with `merge()`.
- If you selected the option to retain the most recently updated data privacy record for merging leads and contacts, but the caller does not have CRUD permission for the selected data privacy record, the merge process selects the data privacy record already associated with the main record.

Redundant relationships

You can't merge accounts or person accounts that are related to the same contact. Remove redundant account-contact relationships before you try to merge accounts.

 **Note:** Starting with API version 15.0, if you specify a value for a field that contains a string, and the value is too big for the field, the call fails and an error is returned. In previous versions of the API the value was truncated and the call succeeded. If you wish to keep the old behavior with versions 15.0 and later, use the [AllowFieldTruncationHeader](#) SOAP header.

Sample Code—Java

This sample merges a victim account with a main account. It creates 2 accounts and attaches a note to the victim. After the merge, the code prints the ID of the victim account and the number of child records updated. In this example, the number of updated records is one, because the note of the merged account is moved to the main account.

```
public Boolean mergeRecords() {
    Boolean success = false;
    // Array to hold the results
    String[] accountIds = new String[2];
    try {
        // Create two accounts to merge
        Account[] accounts = new Account[2];
        Account masterAccount = new Account();
        masterAccount.setName("MasterAccount");
        masterAccount.setDescription("The Account record to merge with.");
        accounts[0] = masterAccount;
```

```

Account accountToMerge = new Account();
accountToMerge.setName("AccountToMerge");
accountToMerge
    .setDescription("The Account record that will be merged.");
accounts[1] = accountToMerge;
SaveResult[] saveResults = connection.create(accounts);

if (saveResults.length > 0) {
    for (int i = 0; i < saveResults.length; i++) {
        if (saveResults[i].isSuccess()) {
            accountIds[i] = saveResults[i].getId();
            System.out.println("Created Account ID: "
                + accountIds[i]);
        } else {
            // If any account is not created,
            // print the error returned and exit
            System.out
                .println("An error occurred while creating account."
                    + " Error message: "
                    + saveResults[i].getErrors()[0].getMessage());
            return success;
        }
    }
}

// Set the Ids of the accounts
masterAccount.setId(accountIds[0]);
accountToMerge.setId(accountIds[1]);

// Attach a note to the account to be merged with the master,
// which will get re-parented after the merge
Note note = new Note();
System.out.println("Attaching note to record " +
    accountIds[1]);
note.setParentId(accountIds[1]);
note.setTitle("Merged Notes");
note.setBody("This note will be moved to the "
    + "MasterAccount during merge");
SaveResult[] sRes = connection.create(new SObject[] { note });
if (sRes[0].isSuccess()) {
    System.out.println("Created Note record.");
} else {
    Error[] errors = sRes[0].getErrors();
    System.out.println("Could not create Note record: "
        + errors[0].getMessage());
}

// Perform the merge
MergeRequest mReq = new MergeRequest();
masterAccount.setDescription("Was merged");
mReq.setMasterRecord(masterAccount);
mReq.setRecordToMergeIds(new String[] { saveResults[1].getId() });
MergeResult mRes = connection.merge(new MergeRequest[] { mReq })[0];

```

```

if (mRes.isSuccess())
{
    System.out.println("Merge successful.");
    // Write the IDs of merged records
    for(String mergedId : mRes.getMergedRecordIds()) {
        System.out.println("Merged Record ID: " + mergedId);
    }
    // Write the updated child records. (In this case the note.)
    System.out.println(
        "Child records updated: " + mRes.getUpdatedRelatedIds().length);
    success = true;
} else {
    System.out.println("Failed to merge records. Error message: " +
        mRes.getErrors()[0].getMessage());
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
return success;
}

```

Sample Code—C#

This sample merges a victim account with a main account. It creates 2 accounts and attaches a note to the victim. After the merge, the code prints the ID of the victim account and the number of child records updated. In this example, the number of updated records is one, because the note of the merged account is moved to the main account.

```

public Boolean mergeRecords()
{
    Boolean success = false;
    // Array to hold the results
    String[] accountIds = new String[2];
    try
    {
        // Create two accounts to merge
        Account[] accounts = new Account[2];
        Account masterAccount = new Account();
        masterAccount.Name = "MasterAccount";
        masterAccount.Description = "The Account record to merge with.";
        accounts[0] = masterAccount;
        Account accountToMerge = new Account();
        accountToMerge.Name = "AccountToMerge";
        accountToMerge
            .Description = "The Account record that will be merged.";
        accounts[1] = accountToMerge;
        SaveResult[] saveResults = binding.create(accounts);

        if (saveResults.Length > 0)
        {
            for (int i = 0; i < saveResults.Length; i++)
            {
                if (saveResults[i].success)

```

```

    {
        accountIds[i] = saveResults[i].id;
        Console.WriteLine("Created Account ID: "
            + accountIds[i]);
    }
    else
    {
        // If any account is not created,
        // print the error returned and exit
        Console.WriteLine("An error occurred while creating account."
            + " Error message: "
            + saveResults[i].errors[0].message);
        return success;
    }
}
}

// Set the Ids of the accounts
masterAccount.Id = accountIds[0];
accountToMerge.Id = accountIds[1];

// Attach a note to the account to be merged with the master,
// which will get re-parented after the merge
Note note = new Note();
Console.WriteLine("Attaching note to record " +
    accountIds[1]);
note.ParentId = accountIds[1];
note.Title = "Merged Notes";
note.Body = "This note will be moved to the "
    + "MasterAccount during merge";
SaveResult[] sRes = binding.create(new sObject[] { note });
if (sRes[0].success)
{
    Console.WriteLine("Created Note record.");
}
else
{
    Error[] errors = sRes[0].errors;
    Console.WriteLine("Could not create Note record: "
        + errors[0].message);
}

// Perform the merge
MergeRequest mReq = new MergeRequest();
masterAccount.Description = "Was merged";
mReq.masterRecord = masterAccount;
mReq.recordToMergeIds = new String[] { saveResults[1].id };

MergeResult mRes = binding.merge(new MergeRequest[] { mReq })[0];

if (mRes.success)
{
    Console.WriteLine("Merge successful.");
    // Write the IDs of merged records

```



```

    foreach (String mergedId in mRes.mergedRecordIds)
    {
        Console.WriteLine("Merged Record ID: " + mergedId);
    }
    // Write the updated child records. (In this case the note.)
    Console.WriteLine(
        "Child records updated: " + mRes.updatedRelatedIds.Length);
    success = true;
}
else
{
    Console.WriteLine("Failed to merge records. Error message: " +
        mRes.errors[0].message);
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
return success;
}

```

Arguments

This call accepts an array of MergeRequest objects. A MergeRequest object contains the following properties.

Name	Type	Description
masterRecord	sObject	Required. Must provide the ID of the object that other records will be merged into. Optionally, provide the fields to be updated and their values.
recordToMergeIds	ID[]	Required. Minimum of one, maximum of two. The other record or records to be merged into the main record.
AdditionalInformationMap	map	A field-value map. <ul style="list-style-type: none"> • Merge a portal user ID: <ul style="list-style-type: none"> - name: PortalUserId - value: ID of the portal user • In all other merge cases, set to null.

Response

[MergeResult\[\]](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)[InvalidIdFault](#)

SEE ALSO:

[API Call Basics](#)

MergeResult

The `merge()` call returns a `MergeResult` object, which has the following properties:

Name	Type	Description
<code>errors</code>	<code>Error[]</code>	If an error occurred during the <code>merge()</code> call, an array of one or more Error objects providing the error code and description.
<code>id</code>	<code>ID</code>	ID of the primary record, the record into which the other records were merged.
<code>mergedRecordIds</code>	<code>ID[]</code>	ID of the records that were merged into the primary record. If successful, the values match <code>mergeRequest.recordToMergeIds</code> .
<code>success</code>	<code>boolean</code>	Indicates whether the merge was successful (<code>true</code>) or not (<code>false</code>).
<code>updatedRelatedIds</code>	<code>ID[]</code>	ID of all related records that were moved (reparented) as a result of the merge, and that are viewable by the user sending the merge call.

performQuickActions()




Executes quick actions of type create or update.

Syntax

```
PerformQuickActionResult[] = connection.performQuickActions(PerformQuickActionRequest
PerformQuickActionRequest[]);
```

Usage

Use the `performQuickActions()` call to perform a specific quick action. Returns an array of `PerformQuickActionResult` objects.

-  **Note:** In API version 46.0 and later, the `apiName` for a global quick action can include the prefix `Global.` in a `performQuickActions()` request body. The request body also accepts global quick action API names without the prefix.
-  **Note:** If you're accessing the API using a custom community URL and you use the `performQuickActions()` call to create a group, the group is only available within that community.
-  **Note:** The `OutgoingEmail` entity can be created only via calls from the `performQuickAction` API.

Sample—Java

This sample uses a quick action to create a new contact.

```
public void example() throws Exception {

    PerformQuickActionRequest req = new PerformQuickActionRequest();

    Contact con = new Contact();
    con.setLastName("Smith");

    req.setQuickActionName("Account.QuickCreateContact");
    req.setParentId("001D000000JSaHa");
    /* For version 29.0 and greater, use setContextId */
    req.setRecords(new SObject[] { con }); //you can only save one record here
    PerformQuickActionResult[] pResult =
        conn.performQuickActions(new PerformQuickActionRequest[] { req });
    for(PerformQuickActionResult pr : pResult) {
        assert pr.getSuccess();
        assert pr.getCreated();
        assert pr.getErrors().length == 0;
        System.out.println("Id of the record created: " + pr.getIds()[0]);
        System.out.println("Id of the feeditem for action performed: " +
            pr.getFeedItemIds()[0]);
    }
}
```

Arguments

Name	Type	Description
quickActions	PerformQuickActionRequest	The action request to perform.

PerformQuickActionRequest

Name	Type	Description
parentOrContextId	ID	<ul style="list-style-type: none"> In API version 28.0, <code>parentId</code> is the ID of the sObject on which to create a record for the request. In API version 29.0 and greater, <code>contextId</code> is the ID of the context on which to create a record for the request.
quickActionName	string	The parent or context sObject and action name—for example, <code>Opportunity.QuickCreateOpp</code> .
records	SObject[]	The record to be created. Only one record can be saved at a time.

Response

[PerformQuickActionResult](#)

PerformQuickActionResult

The [performQuickActions\(\)](#) call returns an array of [PerformQuickActionResult](#) objects.

Name	Type	Description
<code>created</code>	boolean	If <code>true</code> , the record was created successfully and if <code>false</code> , no record was created.
<code>errors</code>	<code>Error[]</code>	If an error occurred during the call, an array of one or more Error objects providing the error information.
<code>feedItemIds</code>	<code>ID[]</code>	Returns an array of unique identifiers of a feed item in the form of a string with IDs; in partner portals, a type with an ID is returned.
<code>ids</code>	<code>ID[]</code>	An array of IDs.
<code>success</code>	boolean	If <code>true</code> , the action executed successfully and if <code>false</code> , the action failed.
<code>successMessage</code>	string	Returns the message that displays to the user upon successful completion of the action.

`process()`

Submits an array of approval process instances for approval, or processes an array of approval process instances to be approved, rejected, or removed. For more information, see “Set Up an Approval Process” in the Salesforce Help.

Syntax

```
ProcessResult = connection.process( processType processRequest[] )
```

`processType` can be either [ProcessSubmitRequest](#) or [ProcessWorkitemRequest](#)


Usage

Use the `process()` call to perform either of the following two tasks:

- Submit an array of objects to the approval process. Objects cannot already be in an approval process when submitted. Use the `ProcessSubmitRequest` signature.
- Process an object that has been submitted to the approval process by performing an approval action (Approve or Reject). Use the `ProcessWorkitemRequest` signature.

Requests are processed and a [ProcessResult](#) is returned with the same process instances as sent in the request.

The failure of a particular record will not cause failure of the entire request.

 **Note:** Because you can fire Apex triggers with this call, you may be updating fields that contain strings.

Starting with API version 15.0, if you specify a value for a field that contains a string, and the value is too big for the field, the call fails and an error is returned. In previous versions of the API the value was truncated and the call succeeded. If you wish to keep the old behavior with versions 15.0 and later, use the [AllowFieldTruncationHeader](#) SOAP header.

Sample Code—Java

This sample accepts the ID of the sObject to process the approval for and an array containing the IDs of the next approvers. It creates a process approval request and submits it for approval. Finally, it parses the results of the `process ()` call.

```
public void processRecords(String id, String[] approverIds) {
    ProcessSubmitRequest request = new ProcessSubmitRequest();
    request.setComments("A comment about this approval.");
    request.setObjectId(id);
    request.setNextApproverIds(approverIds);
    try {
        ProcessResult[] processResults = connection
            .process(new ProcessSubmitRequest[] { request });
        for (ProcessResult processResult : processResults) {
            if (processResult.isSuccess()) {
                System.out.println("Approval submitted for: " + id + ":");
                for (int i = 0; i < approverIds.length; i++) {
                    System.out
                        .println("\tBy: " + approverIds[i] + " successful.");
                }
                System.out.println("Process Instance Status: "
                    + processResult.getInstanceStatus());
            } else {
                System.out.println("Approval submitted for: " + id
                    + ", approverIds: " + approverIds.toString() + " FAILED.");
                System.out.println("Error: "
                    + processResult.getErrors().toString());
            }
        }
    }
    catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample accepts the ID of the sObject to process the approval for and an array containing the IDs of the next approvers. It creates a process approval request and submits it for approval. Finally, it parses the results of the `process ()` call.

```
public void processRecords(String id, String[] approverIds)
{
    ProcessSubmitRequest request = new ProcessSubmitRequest();
    request.comments = "A comment about this approval.";
    request.objectId = id;
    request.nextApproverIds = approverIds;
    try
    {
```

```

ProcessResult[] processResults = binding.process(
    new ProcessSubmitRequest[] { request });
foreach (ProcessResult processResult in processResults)
{
    if (processResult.success)
    {
        Console.WriteLine("Approval submitted for: " + id + ":");
        for (int i = 0; i < approverIds.Length; i++)
        {
            Console.WriteLine("\tBy: " + approverIds[i] + " successful.");
        }
        Console.WriteLine("Process Instance Status: "
            + processResult.instanceStatus);
    }
    else
    {
        Console.WriteLine("Approval submitted for: " + id
            + ", approverIds: " + approverIds.ToString() + " FAILED.");
        Console.WriteLine("Error: "
            + processResult.errors.ToString());
    }
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

ProcessSubmitRequest Arguments

Name	Type	Description
comments	string	Text that you want to accompany the submission. Don't reference merge fields or formula expressions. Submission comments appear in the approval history for the specified record. This text also appears in the initial approval request email if the template uses the <code>{!ApprovalRequest.Comments}</code> merge field.
nextApproverIds	ID	If the process requires specification of the next approval, the ID of the user to be assigned the next request.
objectId	ID	The record to submit for approval.
processDefinitionNameOrId	string	The unique name or ID of the specific approval process to which you want the record to be submitted. The process must have the same object type as the record you specified in <code>objectId</code> . Required if <code>skipEntryCriteria</code> is <code>true</code> .

Name	Type	Description
skipEntryCriteria	boolean	If <i>true</i> , the record isn't evaluated against the entry criteria set on the process that is defined in <code>processDefinitionNameOrId</code> .
submitterId	ID	The ID for the user who submitted the record for approval. The user receives notifications about responses to the approval request. The user must be one of the allowed submitters for the process.

ProcessWorkitemRequest Arguments

Name	Type	Description
action	string	For processing an item after being submitted for approval, a string representing the kind of action to take: Approve, Reject, or Remove. Only system administrators can specify Remove. If the Allow submitters to recall approval requests option is selected for the approval process, the submitter can also specify Remove.
comments	string	Text that you want to accompany the submission. Don't reference merge fields or formula expressions. Submission comments appear in the approval history for the specified record. This text also appears in the initial approval request email if the template uses the <code>{!ApprovalRequest.Comments}</code> merge field.
nextApproverIds	ID	If the process requires specification of the next approval, the ID of the user to be assigned the next request.
workitemId	ID	The ID of the ProcessInstanceWorkitem that is being approved, rejected, or removed.

Response

[ProcessResult\[\]](#)

Faults

[ALREADY_IN_PROCESS](#)

[NO_APPLICABLE_PROCESS](#)

SEE ALSO:

[API Call Basics](#)

ProcessResult

The `process()` call returns a `ProcessResult` object, which has the following properties, depending on the type of call (submit for approval or process object already submitted to for approval):

Name	Type	Description
actorIds	ID	IDs of the users who are currently assigned to this approval step.
entityId	ID	The object being processed.
errors	Error[]	The set of errors returned if the request failed.
instanceId	ID	The ID of the ProcessInstance associated with the object submitted for processing.
instanceStatus	string	The status of the current process instance (not an individual object but the entire process instance). The valid values are "Approved," "Rejected," "Removed," or "Pending."
newWorkItemIds	ID[]	Case-insensitive IDs that point to ProcessInstanceWorkitem items (the set of pending approval requests).
success	boolean	true if processing or approval completed successfully.

query ()

Executes a query against the specified object and returns data that matches the specified criteria.

Syntax

```
QueryResult = connection.query(string queryString);
```


Usage

Use the `query ()` call to retrieve data from an object. When a client application invokes the `query ()` call, it passes in a query expression that specifies the object to query, the fields to retrieve, and any conditions that determine whether a given object qualifies. For an extensive discussion about the syntax and rules used for queries, see the [Salesforce SOQL and SOSL Reference Guide](#).

Upon invocation, the API executes the query against the specified object, caches the results of the query on the API, and returns a query response object to the client application. The client application can then use methods on the query response object to iterate through rows in the query response and retrieve information.

Your client application must be logged in with sufficient access rights to query individual objects within the specified object and to query the fields in the specified field list. For more information, see [Factors that Affect Data Access](#).

Certain objects cannot be queried via the API. To query an object via the `query ()` call, its object must be configured as queryable. To determine whether an object can be queried, your client application can invoke the `describeObjects ()` call on the object and inspect its `queryable` property.


 **Tip:** If you use the enterprise WSDL, don't use `describe` to populate a select list. For example, if a Salesforce admin adds a field to the sObject after you consume it, the `describe` call will pull down the field but your toolkit won't know how to serialize it, and your integration may fail.

You can use `queryAll ()` to query on all [Task](#) and [Event](#) records, archived or not. You can also filter on the `isArchived` field to find only the archived objects. You cannot use `query ()`, it automatically filters out all records where `isArchived` is set to `true`. You can insert, update, or delete archived records.

The query result object contains up to 2,000 rows of data by default. If the query results exceed the default, then the client application uses the `queryMore()` call and a server-side cursor to retrieve additional rows in batches. You can adjust the default results batch in the `QueryOptions` on page 356 header. For more information see [Change the Batch Size in Queries](#) in the *SOQL and SOSL Reference*.

Queries that take longer than two minutes to process time out. For timed out queries, the API returns an API fault element of `InvalidQueryLocatorFault`. If a timeout occurs, refactor your query to return or scan a smaller amount of data.

When querying for fields of type Base64 (see [base64](#)), the query response object returns only one record at a time. You can't alter this by changing the batch size of the `query()` call.

 **Note:** For multicurrency organizations, special handling is required when querying currency fields containing values in different currencies. For example, if a client application is querying `PricebookEntry` objects based on values in the `UnitPrice` field, and if the `UnitPrice` amounts are expressed in different currencies, then the query logic must handle this case correctly. For example, if the query is trying to retrieve the product codes of all products with a unit price greater than or equal to \$10 USD, the query expression might look something like this:

```
SELECT Product2Id,ProductCode,UnitPrice FROM PricebookEntry
WHERE (UnitPrice >= 10 and CurrencyIsoCode='USD')
OR (UnitPrice >= 5.47 and CurrencyIsoCode='GBP')
OR (UnitPrice >= 8.19 and CurrencyIsoCode='EUR')
```

Sample Code—Java

This sample executes a query that fetches the first names and last names of all contacts. It calls `query()` with the query string to get the first batch of records. It then calls `queryMore()` in a loop to get subsequent batches of records until no records are returned. It writes the first and last names of the contacts queried to the console.

```
public void queryRecords() {
    QueryResult qResult = null;
    try {
        String soqlQuery = "SELECT FirstName, LastName FROM Contact";
        qResult = connection.query(soqlQuery);
        boolean done = false;
        if (qResult.getSize() > 0) {
            System.out.println("Logged-in user can see a total of "
                + qResult.getSize() + " contact records.");
            while (!done) {
                SObject[] records = qResult.getRecords();
                for (int i = 0; i < records.length; ++i) {
                    Contact con = (Contact) records[i];
                    String fName = con.getFirstName();
                    String lName = con.getLastName();
                    if (fName == null) {
                        System.out.println("Contact " + (i + 1) + ": " + lName);
                    } else {
                        System.out.println("Contact " + (i + 1) + ": " + fName
                            + " " + lName);
                    }
                }
            }
            if (qResult.isDone()) {
                done = true;
            } else {
                qResult = connection.queryMore(qResult.getQueryLocator());
            }
        }
    }
}
```

```

    }
} else {
    System.out.println("No records found.");
}
System.out.println("\nQuery successfully executed.");
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

Sample Code—C#

This sample executes a query that fetches the first names and last names of all contacts. It calls `query()` with the query string to get the first batch of records. It then calls `queryMore()` in a loop to get subsequent batches of records until no records are returned. It writes the first and last names of the contacts queried to the console.

```

public void queryRecords()
{
    QueryResult qResult = null;
    try
    {
        String soqlQuery = "SELECT FirstName, LastName FROM Contact";
        qResult = binding.query(soqlQuery);
        Boolean done = false;
        if (qResult.size > 0)
        {
            Console.WriteLine("Logged-in user can see a total of "
                + qResult.size + " contact records.");
            while (!done)
            {
                sObject[] records = qResult.records;
                for (int i = 0; i < records.Length; ++i)
                {
                    Contact con = (Contact)records[i];
                    String fName = con.FirstName;
                    String lName = con.LastName;
                    if (fName == null)
                    {
                        Console.WriteLine("Contact " + (i + 1) + ": " + lName);
                    }
                    else
                    {
                        Console.WriteLine("Contact " + (i + 1) + ": " + fName
                            + " " + lName);
                    }
                }
            }
            if (qResult.done)
            {
                done = true;
            }
            else
            {
                qResult = binding.queryMore(qResult.queryLocator);
            }
        }
    }
}

```

```

        }
    }
}
else
{
    Console.WriteLine("No records found.");
}
Console.WriteLine("\nQuery succesfully executed.");
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}
}

```

Arguments

Name	Type	Description
queryString	string	Query string that specifies the object to query, the fields to return, and any conditions for including a specific object in the query. For more information, see the Salesforce SOQL and SOSL Reference Guide .

Response

[QueryResult](#)

Faults

[MalformedQueryFault](#)

[InvalidSObjectFault](#)

[InvalidFieldFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[queryAll\(\)](#)

[queryMore\(\)](#)

[API Call Basics](#)

[Change the Batch Size in Queries](#)

QueryResult

The `query()` call returns a `QueryResult` object, which has the following properties:

Name	Type	Description
queryLocator	QueryLocator	A specialized string, similar to ID. Used in queryMore() for retrieving subsequent sets of objects from the query results, if applicable. Represents a server-side cursor. Each user can have up to ten query cursors open at a time.
done	boolean	Indicates whether additional rows need to be retrieved from the query results (<code>false</code>) using queryMore() , or not (<code>true</code>). Your client application can use this value as a loop condition while iterating through the query results.
records	sObject[]	Array of sObjects representing individual objects of the specified object and containing data defined in the field list specified in the queryString . For information on queries that use a <code>GROUP BY</code> clause, see AggregateResult .
size	int	Your client application can use this value to determine whether the query retrieved any rows (<code>size > 0</code>) or not (<code>size = 0</code>). Total number of rows retrieved in the query.

AggregateResult

This object contains the results returned by a [query\(\)](#) if the query contains an aggregate function, such as `MAX()`. `AggregateResult` is an [sObject](#), but unlike other `sObject` objects such as `Contact`, it is read-only and it is only used for query results.

The [QueryResult](#) object has a `records` field that is an array of `sObject` records matching your query. For example, the following query returns an array of `Contact` records in the `records` field.

```
SELECT Id, LastName
FROM Contact
WHERE FirstName = 'Bob'
```

When a SOQL query contains an aggregate function, the results are a set of aggregated data instead of an array of records for a standard object, such as `Contact`. Therefore, the `records` field returns an array of `AggregateResult` records.

For more information on aggregate functions, see "Aggregate Functions" in the [Salesforce SOQL and SOSL Reference Guide](#).

Fields

Each `AggregateResult` object contains a separate field for each of the items in the `SELECT` list. For the enterprise WSDL, retrieve the result for each item by calling `getField()` on an `AggregateResult` object when using WSC client framework. For the partner WSDL, retrieve the result for each item by calling `getField()` on an `sObject` object.

See [Sample Code—Java](#) and [Sample Code—C#](#) for examples that work with the enterprise WSDL.

Sample Code—Java

```
public void queryAggregateResult() {
    try {
        String groupByQuery = "SELECT Account.Name n, " +
            "MAX(Amount) max, MIN(Amount) min " +
            "FROM Opportunity GROUP BY Account.Name";
        QueryResult qr = connection.query(groupByQuery);
        if (qr.getSize() > 0) {
            System.out.println("Query returned " +
                qr.getRecords().length + " results."
            );
        }
    }
}
```

```

);
for (SObject sObj : qr.getRecords()) {
    AggregateResult result = (AggregateResult) sObj;
    System.out.println("aggResult.Account.Name: " +
        result.getField("n")
    );
    System.out.println("aggResult.max: " +
        result.getField("max")
    );
    System.out.println("aggResult.min: " +
        result.getField("min")
    );
    System.out.println();
}
} else {
    System.out.println("No results found.");
}
System.out.println("\nQuery successfully executed.");
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

Sample Code—C#

```

private void testAggregateResult()
{
    try
    {
        QueryResult qr = null;

        binding.QueryOptionsValue = new QueryOptions();

        String sqlStr = "SELECT Name, " +
            "MAX(Amount), " +
            "MIN(Amount) " +
            "FROM Opportunity " +
            "GROUP BY Name";

        qr = binding.query(sqlStr);

        if (qr.size > 0)
        {
            for (int i = 0; i < qr.records.Length; i++)
            {

                sforce.AggregateResult ar = (AggregateResult)qr.records[i];

                foreach (XmlElement e in ar.Any)
                    Console.WriteLine(
                        "{0} - {1}",
                        e.LocalName,

```

```


                e.InnerText
            );
        }
    }
    else
    {
        Console.WriteLine("No records found");
    }
    Console.WriteLine("Query successfully executed.");
}
catch (Exception ex)
{
    Console.WriteLine(
        "\nFailed to execute query successfully." +
        "error message was: \n" +
        ex.Message
    );
}
}
}

```

QueryLocator


In the `QueryResult` object returned by the `query()` call, `queryLocator` contains a value that you use in the subsequent `queryMore()` call. Note the following guidelines:

- Use a given `queryLocator` value only one time. When you pass it in a `queryMore()` call, the API returns a new `queryLocator` in the `QueryResult`.
- `QueryLocator` objects expire automatically after 15 minutes of inactivity.
- A user can have up to 10 query cursors open at a time. If 10 `QueryLocator` cursors are open when a client application, logged in as the same user, attempts to open a new one, then the oldest of the 10 cursors is released. If the client application attempts to open the released query cursor, an error results.

 **Note:** Cursor limits for different Lightning Platform features are tracked separately. For example, you can have 10 SOAP API query cursors, 10 Metadata API cursors, and 50 Apex query cursors open at the same time.

A `QueryLocator` represents a server-side cursor.

queryAll()

 **[other]:** Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

Retrieves data from specified objects, whether or not they have been deleted.

Syntax

```
QueryResult = connection.queryAll(string queryString);
```

Usage

Use `queryAll` to identify the records that have been deleted because of a merge or delete. `queryAll` has read-only access to the field `isDeleted`; otherwise it is the same as `query()`.

To find records that have been deleted (in preparation for undeleting them with the `undelete()` call), specify `isDeleted = true` in the query string, and for merged records, request the `masterRecord`. For example:

```
SELECT id, isDeleted, masterRecordId FROM Account WHERE masterRecordId='100000000000Abc'
```

You can use `queryAll()` to query on all `Task` and `Event` records, archived or not. You can also filter on the `isArchived` field to find only the archived objects. You cannot use `query()` as it automatically filters out all records where `isArchived` is set to `true`. You can update or delete archived records, though you cannot update the `isArchived` field. If you use the API to insert activities that meet the criteria listed below, the activities will be archived during the next run of the archival background process.

Because Salesforce doesn't track changes to external data, `queryAll()` behaves the same as `query()` for external objects.

For additional information about using `queryAll`, see `query()`.

Sample Code—Java

This sample performs a query to get all the accounts, whether they're deleted or not. It sets a custom batch size of 250 records. It fetches all batches of records by calling `queryAll()` the first time and then `queryMore()`. The names and the value of the `isDeleted` fields of all returned accounts are written to the console.

```
public void queryAllRecords() {
    // Setting custom batch size
    connection.setQueryOptions(250);

    try {
        String soqlQuery = "SELECT Name, IsDeleted FROM Account";
        QueryResult qr = connection.queryAll(soqlQuery);
        boolean done = false;
        if (qr.getSize() > 0) {
            System.out.println("Logged-in user can see a total of "
                + qr.getSize()
                + " contact records (including deleted records).");
            while (!done) {
                SObject[] records = qr.getRecords();
                for (int i = 0; i < records.length; i++) {
                    Account account = (Account) records[i];
                    boolean isDel = account.getIsDeleted();
                    System.out.println("Account " + (i + 1) + ": "
                        + account.getName() + " isDeleted = "
                        + account.getIsDeleted());
                }
                if (qr.isDone()) {
                    done = true;
                } else {
                    qr = connection.queryMore(qr.getQueryLocator());
                }
            }
        } else {
            System.out.println("No records found.");
        }
    }
}
```

```

    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}

```

Sample Code—C#

This sample performs a query to get all the accounts, whether they're deleted or not. It sets a custom batch size of 250 records. It fetches all batches of records by calling `queryAll()` the first time and then `queryMore()`. The names and the value of the `isDeleted` fields of all returned accounts are written to the console.

```

public void queryAllRecords()
{
    // Setting custom batch size
    QueryOptions qo = new QueryOptions();
    qo.batchSize = 250;
    qo.batchSizeSpecified = true;
    binding.QueryOptionsValue = qo;

    try
    {
        String sqlQuery = "SELECT Name, IsDeleted FROM Account";
        QueryResult qr = binding.queryAll(sqlQuery);
        Boolean done = false;
        if (qr.size > 0)
        {
            Console.WriteLine("Logged-in user can see a total of "
                + qr.size
                + " contact records (including deleted records).");
            while (!done)
            {
                sObject[] records = qr.records;
                for (int i = 0; i < records.Length; i++)
                {
                    Account account = (Account)records[i];
                    Boolean isDel = (Boolean)account.IsDeleted;
                    Console.WriteLine("Account " + (i + 1) + ": "
                        + account.Name + " isDeleted = "
                        + account.IsDeleted);
                }
                if (qr.done)
                {
                    done = true;
                }
                else
                {
                    qr = binding.queryMore(qr.queryLocator);
                }
            }
        }
        else
        {
            Console.WriteLine("No records found.");
        }
    }
}

```



```

    }
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}
}

```

Arguments

Name	Type	Description
queryString	string	Query string that specifies the object to query, the fields to return, and any conditions for including a specific object in the query. For more information, see the Salesforce SOQL and SOSL Reference Guide .

Response

[QueryResult](#)

Faults

[MalformedQueryFault](#)

[InvalidObjectFault](#)

[InvalidFieldFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[API Call Basics](#)

[queryMore\(\)](#)

queryMore ()

Retrieves the next batch of objects from a [query \(\)](#).

Syntax


```
QueryResult = connection.queryMore( QueryLocator QueryLocator);
```

Usage

Use this call to process [query \(\)](#) calls that retrieve a large number of records. The [query \(\)](#) call retrieves up to 2,000 record in the first batch returned and creates a server-side cursor that is represented in the [queryLocator](#) object. The [queryMore \(\)](#) call processes

subsequent records in additional batches, resets the server-side cursor, and returns a newly generated [QueryLocator](#). To iterate through records in the result set, you generally call `queryMore()` repeatedly until all records in the result set have been processed (the `Done` flag is `true`). See [Change the Batch Size in Queries in the Salesforce SOQL and SOSL Reference Guide](#) for more information.

You can't use `queryMore()` if a query includes a `GROUP BY` clause. See `GROUP BY` in the [Salesforce SOQL and SOSL Reference Guide](#) for more information.

 **Note:** A `queryMore()` call on a parent object invalidates all child cursors in the previous result set. If you need the results from the child, you must use `queryMore()` on those results before using `queryMore()` on the parent results.

When querying external objects, Salesforce Connect accesses the external data in real time via Web service callouts. Each `queryMore()` call results in a Web service callout. The batch boundaries and page sizes depend on your adapter and how you set up the external data source.

We recommend the following:

- When possible, avoid paging by filtering your queries of external objects to return fewer rows than the batch size, which by default is 2,000 rows. Remember, obtaining each batch requires a `queryMore()` call, which results in a Web service callout.
- If the external data frequently changes, avoid using `queryMore()` calls. If the external data is modified between `queryMore()` calls, you can get an unexpected `QueryResult`.

If the primary or “driving” object for a `SELECT` statement is an external object, `queryMore()` supports only that primary object and doesn't support subqueries.

By default, the OData 2.0 and 4.0 adapters for Salesforce Connect use client-driven paging. With client-driven paging, OData adapters convert each `queryMore()` call into an OData query that uses the `$skip` and `$top` system query options to specify the batch boundary and page size. These options are similar to using `LIMIT` and `OFFSET` clauses to page through a result set. If you enable server-driven paging on an external data source, Salesforce ignores the requested page sizes, including the default `queryMore()` batch size of 2,000 rows. The pages returned by the external system determine the batches, but each page can't exceed 2,000 rows.

Sample Code—Java

This sample executes a query that fetches the first names and last names of all contacts. It calls `query()` with the query string to get the first batch of records. It then calls `queryMore()` in a loop to get subsequent batches of records until no records are returned. It writes the first and last names of the contacts queried to the console.

```
public void queryRecords() {
    QueryResult qResult = null;
    try {
        String soqlQuery = "SELECT FirstName, LastName FROM Contact";
        qResult = connection.query(soqlQuery);
        boolean done = false;
        if (qResult.getSize() > 0) {
            System.out.println("Logged-in user can see a total of "
                + qResult.getSize() + " contact records.");
            while (!done) {
                SObject[] records = qResult.getRecords();
                for (int i = 0; i < records.length; ++i) {
                    Contact con = (Contact) records[i];
                    String fName = con.getFirstName();
                    String lName = con.getLastName();
                    if (fName == null) {
                        System.out.println("Contact " + (i + 1) + ": " + lName);
                    } else {
                        System.out.println("Contact " + (i + 1) + ": " + fName);
                    }
                }
            }
        }
    }
}
```

```

        + " " + lName);
    }
}
if (qResult.isDone()) {
    done = true;
} else {
    qResult = connection.queryMore(qResult.getQueryLocator());
}
}
} else {
    System.out.println("No records found.");
}
System.out.println("\nQuery successfully executed.");
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

Sample Code—C#

This sample executes a query that fetches the first names and last names of all contacts. It calls `query()` with the query string to get the first batch of records. It then calls `queryMore()` in a loop to get subsequent batches of records until no records are returned. It writes the first and last names of the contacts queried to the console.

```

public void queryRecords()
{
    QueryResult qResult = null;
    try
    {
        String sqlQuery = "SELECT FirstName, LastName FROM Contact";
        qResult = binding.query(sqlQuery);
        Boolean done = false;
        if (qResult.size > 0)
        {
            Console.WriteLine("Logged-in user can see a total of "
                + qResult.size + " contact records.");
            while (!done)
            {
                sObject[] records = qResult.records;
                for (int i = 0; i < records.Length; ++i)
                {
                    Contact con = (Contact)records[i];
                    String fName = con.FirstName;
                    String lName = con.LastName;
                    if (fName == null)
                    {
                        Console.WriteLine("Contact " + (i + 1) + ": " + lName);
                    }
                    else
                    {
                        Console.WriteLine("Contact " + (i + 1) + ": " + fName
                            + " " + lName);
                    }
                }
            }
        }
    }
}

```

```

    }
    if (qResult.done)
    {
        done = true;
    }
    else
    {
        qResult = binding.queryMore(qResult.queryLocator);
    }
}
}
else
{
    Console.WriteLine("No records found.");
}
Console.WriteLine("\nQuery succesfully executed.");
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

Arguments

Name	Type	Description
queryLocator	QueryLocator	Represents the server-side cursor that tracks the current processing location in the query result set.

Response

[QueryResult](#)

Faults

[InvalidQueryLocatorFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[query\(\)](#)


[API Call Basics](#)

[Change the Batch Size in Queries](#)

QueryResult

The `queryMore()` call returns a `QueryResult` object, which has the following properties:

Name	Type	Description
<code>queryLocator</code>	<code>QueryLocator</code>	A specialized string, similar to ID. Used in the subsequent <code>queryMore()</code> call for retrieving sets of objects from the query results, if applicable.
<code>done</code>	boolean	Indicates whether additional rows need to be retrieved from the query results (<code>false</code>) using another <code>queryMore()</code> call, or not (<code>true</code>). Your client application can use this value as a loop condition while iterating through the query results.
<code>records</code>	<code>sObject[]</code>	Array of <code>sObjects</code> representing individual objects of the specified object and containing data defined in the field list specified in the <code>queryString</code> .
<code>size</code>	int	Total number of rows retrieved in the query. Your client application can use this value to determine whether the query retrieved any rows (<code>size != 0</code>) or not (<code>size = 0</code>). When querying external objects, the system may not know the number of rows that are retrieved from the external data source. If this situation occurs, <code>size = -1</code> .

 **Note:** A `queryMore()` call on a parent object invalidates all child cursors in the previous result set. If you need the results from the child, you must use `queryMore()` on those results before using `queryMore()` on the parent results.

QueryLocator

In the `QueryResult` object returned by the `queryMore()` call, `queryLocator` contains a value that you will use in the subsequent `queryMore()` call. Note the following guidelines for using this value:

- Use a `queryLocator` only once. When you pass it in a `queryMore()` call, the API returns a new `queryLocator` in the `QueryResult`.
- The `queryLocator` value expires automatically after 15 minutes of inactivity.
- A user can have up to ten query cursors open at a time. If ten `QueryLocator` cursors are opened when a client application with the same logged-in user attempts to open a new cursor, then the oldest of the ten cursors is released.
- You can't use a custom metadata query as a `queryLocator`.

A `QueryLocator` represents a server-side cursor.

 **Note:** A `queryMore()` call on a parent object invalidates all child cursors in the previous result set. If you need the results from the child, you must use `queryMore()` on those results before using `queryMore()` on the parent results.

retrieve()

Retrieves one or more records based on the specified IDs.

Syntax

```
sObject[] result = connection.retrieve(string fieldList, string sObjectType, ID ids[]);
```

Usage

Use the `retrieve()` call to retrieve individual records from an object. The client application passes the list of fields to retrieve, the object, and an array of record IDs to retrieve. The `retrieve()` call does not return records that have been deleted.

In general, you use `retrieve()` when you know in advance the IDs of the records to retrieve. Use `query()` instead to obtain records when you do not know the IDs or when you want to specify other selection criteria.

Client applications can use `retrieve()` to perform a client-side join. For example, a client application can run a `query()` to obtain a set of [Opportunity](#) records, iterate through the returned opportunity records, obtain the `accountId` for each opportunity, and then call `retrieve()` to obtain [Account](#) information for those `accountIds`.

Records for certain objects cannot be retrieved via the API. To retrieve a record via the `retrieve()` call, its object must be configured as retrieveable (`retrieveable` is `true`). To determine whether an object can be retrieved, your client application can invoke the [describeSObjects\(\)](#) call on the object and inspect its `retrieveable` property.

Your client application must be logged in with sufficient access rights to retrieve records within the specified object and to retrieve the fields in the specified field list. For more information, see [Factors that Affect Data Access](#).

Sample Code—Java

This sample retrieves the Id, Name, and Website of the specified Account records. It writes the fields of the retrieved records to the console.

```
public void retrieveRecords(String[] ids) {
    try {
        SObject[] sObjects = connection.retrieve("ID, Name, Website",
            "Account", ids);
        // Verify that some objects were returned.
        // Even though we began with valid object IDs,
        // someone else might have deleted them in the meantime.
        if (sObjects != null) {
            for (int i = 0; i < sObjects.length; i++) {
                // Cast the SObject into an Account object
                Account retrievedAccount = (Account) sObjects[i];
                if (retrievedAccount != null) {
                    System.out.println("Account ID: " + retrievedAccount.getId());
                    System.out.println("Account Name: " + retrievedAccount.getName());
                    System.out.println("Account Website: "
                        + retrievedAccount.getWebsite());
                }
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample retrieves the Id, Name, and Website of the specified Account records. It writes the fields of the retrieved records to the console.

```
public void retrieveRecords(String[] ids)
{
    try
    {
        sObject[] sObjects = binding.retrieve("ID, Name, Website",
            "Account", ids);
        // Verify that some objects were returned.
        // Even though we began with valid object IDs,
        // someone else might have deleted them in the meantime.
        if (sObjects != null)
        {
            for (int i = 0; i < sObjects.Length; i++)
            {
                // Cast the SObject into an Account object
                Account retrievedAccount = (Account)sObjects[i];
                if (retrievedAccount != null)
                {
                    Console.WriteLine("Account ID: " + retrievedAccount.Id);
                    Console.WriteLine("Account Name: " + retrievedAccount.Name);
                    Console.WriteLine("Account Website: "
                        + retrievedAccount.Website);
                }
            }
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

Arguments

Name	Type	Description
fieldList	string	List of one or more fields in the specified object, separated by commas. You must specify valid field names and must have read-level permissions to each specified field. The fieldList defines the ordering of fields in the result .
sObjectType	string	Object from which to retrieve data. The specified value must be a valid object for your organization. For a complete list of objects, see Standard Objects .
ids	ID[]	Array of one or more IDs of the objects to retrieve. You can pass a maximum of 2000 object IDs to the <code>retrieve()</code> call. For information on IDs, see ID Field Type .

Response

Name	Type	Description
result	sObject[]	Array of one or more sObjects representing individual records of the specified object. The number of sObject returned in the array matches the number of IDs passed into the <code>retrieve()</code> call. If you do not have access to an object or if a passed ID is invalid, the array returns <code>null</code> for that object. For information on IDs, see ID Field Type .

Faults

[InvalidSObjectFault](#)

[InvalidFieldFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[API Call Basics](#)

search ()

Executes a text search in your organization's data.

Syntax

```
SearchResult = connection.search(String searchString);
```

Usage

Use [search\(\)](#) to search for records based on a search string. The search call supports searching custom objects. For an extensive discussion about the syntax and rules used for text searches, see the [Salesforce SOQL and SOSL Reference Guide](#).

Certain objects cannot be searched via the API, such as [Attachment](#) objects. To search an object via the `search()` call, the object must be configured as searchable (`isSearchable` is `true`). To determine whether an object can be searched, your client application can invoke the [describeObjects\(\)](#) call on the object and inspect its `searchable` property.

Sample Code—Java

This sample makes the `search()` call by passing it a SOSL query, which returns contacts, leads, and accounts whose phone fields contain a specified value. Next, it gets the sObject records from the results and stores the records in arrays depending on the record type. Finally, it writes the fields of the returned contacts, leads, and accounts to the console.

```
public void searchSample() {
    try {
        // Perform the search using the SOSL query.
        SearchResult sr = connection.search(
            "FIND {4159017000} IN Phone FIELDS RETURNING "
```



```
+ "Contact(Id, Phone, FirstName, LastName), "  
+ "Lead(Id, Phone, FirstName, LastName), "  
+ "Account(Id, Phone, Name)");  
  
// Get the records from the search results.  
SearchRecord[] records = sr.getSearchRecords();  
  
ArrayList<Contact> contacts = new ArrayList<Contact>();  
ArrayList<Lead> leads = new ArrayList<Lead>();  
ArrayList<Account> accounts = new ArrayList<Account>();  
  
// For each record returned, find out if it's a  
// contact, lead, or account and add it to the  
// appropriate array, then write the records  
// to the console.  
if (records.length > 0) {  
    for (int i = 0; i < records.length; i++) {  
        SObject record = records[i].getRecord();  
        if (record instanceof Contact) {  
            contacts.add((Contact) record);  
        } else if (record instanceof Lead) {  
            leads.add((Lead) record);  
        } else if (record instanceof Account) {  
            accounts.add((Account) record);  
        }  
    }  
}  
  
System.out.println("Found " + contacts.size() + " contacts.");  
for (Contact c : contacts) {  
    System.out.println(c.getId() + ", " + c.getFirstName() + ", "  
+ c.getLastName() + ", " + c.getPhone());  
}  
System.out.println("Found " + leads.size() + " leads.");  
for (Lead d : leads) {  
    System.out.println(d.getId() + ", " + d.getFirstName() + ", "  
+ d.getLastName() + ", " + d.getPhone());  
}  
System.out.println("Found " + accounts.size() + " accounts.");  
for (Account a : accounts) {  
    System.out.println(a.getId() + ", " + a.getName() + ", "  
+ a.getPhone());  
}  
} else {  
    System.out.println("No records were found for the search.");  
}  
} catch (Exception ce) {  
    ce.printStackTrace();  
}  
}
```

Sample Code—C#

This sample makes the `search()` call by passing it a SOSL query, which returns contacts, leads, and accounts whose phone fields contain a specified value. Next, it gets the sObject records from the results and stores the records in arrays depending on the record type. Finally, it writes the fields of the returned contacts, leads, and accounts to the console.

```
public void searchSample()
{
    try
    {
        // Perform the search using the SOSL query.
        SearchResult sr = binding.search(
            "FIND {4159017000} IN Phone FIELDS RETURNING "
            + "Contact(Id, Phone, FirstName, LastName), "
            + "Lead(Id, Phone, FirstName, LastName), "
            + "Account(Id, Phone, Name)");

        // Get the records from the search results.
        SearchRecord[] records = sr.searchRecords;

        List<Contact> contacts = new List<Contact>();
        List<Lead> leads = new List<Lead>();
        List<Account> accounts = new List<Account>();

        // For each record returned, find out if it's a
        // contact, lead, or account and add it to the
        // appropriate array, then write the records
        // to the console.
        if (records.Length > 0)
        {
            for (int i = 0; i < records.Length; i++)
            {
                sObject record = records[i].record;
                if (record is Contact)
                {
                    contacts.Add((Contact)record);
                }
                else if (record is Lead)
                {
                    leads.Add((Lead)record);
                }
                else if (record is Account)
                {
                    accounts.Add((Account)record);
                }
            }

            Console.WriteLine("Found " + contacts.Count + " contacts.");
            foreach (Contact c in contacts)
            {
                Console.WriteLine(c.Id + ", " +
                    c.FirstName + ", " +
                    c.LastName + ", " +
                    c.Phone);
            }
        }
    }
}
```

```

    }
    Console.WriteLine("Found " + leads.Count + " leads.");
    foreach (Lead d in leads)
    {
        Console.WriteLine(d.Id + ", " +
            d.FirstName + ", " +
            d.LastName + ", " +
            d.Phone);
    }
    Console.WriteLine("Found " + accounts.Count + " accounts.");
    foreach (Account a in accounts)
    {
        Console.WriteLine(a.Id + ", " +
            a.Name + ", " +
            a.Phone);
    }
}
else
{
    Console.WriteLine("No records were found for the search.");
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}
}

```

Arguments

Name	Type	Description
search	string	Search string that specifies the text expression to search for, the scope of fields to search, the list of objects and fields to retrieve, and the maximum number of records to return. For more information, see the Salesforce SOQL and SOSL Reference Guide .

Response

[SearchResult](#)

Fault

[InvalidFieldFault](#)

[InvalidObjectFault](#)

[MalformedSearchFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[API Call Basics](#)

SearchResult

The `search()` call returns a `SearchResult` object, which has the following properties.

Name	Type	Description
<code>queryId</code>	string	Unique identifier for the SOSP search.
<code>searchRecords</code>	SearchRecord[]	Array of <code>SearchRecord</code> objects, each of which contains an <code>sObject</code> .
<code>searchResultsMetadata</code>	SearchResultsMetadata	Metadata for <code>SearchRecords</code> .

SearchRecord

Represents an individual record returned from a search.

Name	Type	Description
<code>record</code>	sObject	The individual record returned by the search.
<code>searchRecordMetadata</code>	SearchRecordMetadata	Metadata for <code>searchRecords</code> .
<code>snippet</code>	SearchSnippet	On the search results page, shows terms that match the search string, highlighted within the surrounding text.

SearchRecordMetadata

Metadata for search results at the record level.

Name	Type	Description
<code>searchPromoted</code>	boolean	Indicates that an article has been promoted in search results. Admins define promoted search terms by adding promoted terms to knowledge articles. Users who search for these keywords see the article first in search results. Available in API version 42.0 and later.
<code>spellCorrected</code>	boolean	Indicates that a record matches a spell-corrected search term. Appears in the response only when true.

SearchSnippet

Excerpts shown on search results pages for article, case, feed, and idea searches.

Name	Type	Description
text	string	The excerpt that contains the match for the search term.
wholeFields	WholeFields	The list of highlighted fields.

WholeFields

Contains the complete text of each field that contains highlighting for terms that match the search query. The highlighted terms are surrounded by <mark> tags.

Name	Type	Description
name	string	The name of the highlighted field.
value	string	The highlighted text.

SearchResultsMetadata

Global metadata for the search result.

Name	Type	Description
entityMetadata	EntitySearchMetadata	Search results metadata at the object level.

EntitySearchMetadata

Metadata for search results at the object level.

Name	Type	Description
fieldMetadata	FieldLevelSearchMetadata	Metadata for search results at the field level.
searchPromotedMetadata	EntitySearchPromotionMetadata	Metadata for search term promotion at the object level. Available in API version 42.0 and later.
spellCorrectionMetadata	EntitySpellCorrectionMetadata	Metadata for spelling correction at the object level.
entityName	string	Identifies the object.

FieldLevelSearchMetadata

Metadata for search results at the field level.

Name	Type	Description
name	string	The field name.
label	string	The field label.
type	string	The field type.

EntitySearchPromotionMetadata

Metadata for search term promotion at the object level. Appears in the response only when at least one article for an object is a promoted result. Available in API version 42.0 and later.

Name	Type	Description
<code>promotedResultCount</code>	<code>int</code>	Count of promoted article results at the object level.

EntitySpellCorrectionMetadata

Metadata for spelling correction at the object level. Appears in the response only when at least one record for an object matches a spell-corrected search term.

Name	Type	Description
<code>correctedQuery</code>	<code>string</code>	The spell-corrected search term.
<code>hasNonCorrectedResults</code>	<code>boolean</code>	If <code>true</code> , indicates that the user has access to at least one record that matches a search term that wasn't spell-corrected. Each object sometimes returns a different value.

SEE ALSO:

[WITH SNIPPET](#)

[WITH SPELL_CORRECTION](#)

undelete ()

Undeletes records from the Recycle Bin.

Syntax

```
UndeleteResult[] = connection.undelete(ID[] ids );
```


Usage

Use this call to restore any deleted record that is undeletable. Undeletable records include those in the Recycle Bin. Records can be put in the Recycle Bin as the result of a [merge \(\)](#) or [delete \(\)](#) call. You can identify deleted records, including records deleted as the result of a merge, using the [queryAll \(\)](#) call.

You should verify that a record can be undeleted before attempting to delete it. Some records cannot be undeleted, for example, [Account](#) records can be undeleted, but not [AccountTeamMember](#) records. To verify that a record can be undeleted, check that the value of the [undeletable](#) flag in the [DescribeSObjectResult](#) for that object is set to `true`.

Since a delete call cascade-deletes child records, an undelete call will undelete the cascade-deleted records. For example, deleting an account will delete all the contacts associated with that account.

You can undelete records that were deleted as the result of a merge, but the child objects will have been re-parented, which cannot be undone.

 **Note:** Starting with API version 15.0, if you specify a value for a field that contains a string, and the value is too big for the field, the call fails and an error is returned. In previous versions of the API the value was truncated and the call succeeded. If you wish to keep the old behavior with versions 15.0 and later, use the [AllowFieldTruncationHeader](#) SOAP header.

This call supports the [AllOrNoneHeader](#), [AllowFieldTruncationHeader](#), and [CallOptions](#) headers.

Rollback on Error

The [AllOrNoneHeader](#) header allows you to roll back all changes unless all records are processed successfully. This header is available in API version 20.0 and later. Allows a call to roll back all changes unless all records are processed successfully.

Sample Code—Java

This sample calls `queryAll()` to get the last five deleted accounts. It then passes the IDs of these accounts to `undelete()`, which restores these accounts. Finally, it checks the results of the call and writes the IDs of the restored accounts or any errors to the console.

```
public void undeleteRecords() {
    try {
        // Get the accounts that were last deleted
        // (up to 5 accounts)
        QueryResult qResult = connection
            .queryAll("SELECT Id, SystemModstamp FROM "
                + "Account WHERE IsDeleted=true "
                + "ORDER BY SystemModstamp DESC LIMIT 5");

        String[] Ids = new String[qResult.getSize()];
        // Get the IDs of the deleted records
        for (int i = 0; i < qResult.getSize(); i++) {
            Ids[i] = qResult.getRecords()[i].getId();
        }

        // Restore the records
        UndeleteResult[] undelResults = connection.undelete(Ids);

        // Check the results
        for (UndeleteResult result : undelResults) {
            if (result.isSuccess()) {
                System.out.println("Undeleted Account ID: " + result.getId());
            } else {
                if (result.getErrors().length > 0) {
                    System.out.println("Error message: "
                        + result.getErrors()[0].getMessage());
                }
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample calls `queryAll()` to get the last five deleted accounts. It then passes the IDs of these accounts to `undelete()`, which restores these accounts. Finally, it checks the results of the call and writes the IDs of the restored accounts or any errors to the console.

```
public void undeleteRecords()
{
    try
    {
        // Get the accounts that were last deleted
        // (up to 5 accounts)
        QueryResult qResult = binding.queryAll(
            "SELECT Id, SystemModstamp FROM " +
            "Account WHERE IsDeleted=true " +
            "ORDER BY SystemModstamp DESC LIMIT 5");

        String[] Ids = new String[qResult.size];
        // Get the IDs of the deleted records
        for (int i = 0; i < qResult.size; i++)
        {
            Ids[i] = qResult.records[i].Id;
        }

        // Restore the records
        UndeleteResult[] undeleteResults = binding.undelete(Ids);

        // Check the results
        foreach (UndeleteResult result in undeleteResults)
        {
            if (result.success)
            {
                Console.WriteLine("Undeleted Account ID: " +
                    result.id);
            }
            else
            {
                if (result.errors.Length > 0)
                {
                    Console.WriteLine("Error message: " +
                        result.errors[0].message);
                }
            }
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```


Arguments

Name	Type	Description
<code>ids</code>	<code>ID[]</code>	IDs of the records to be restored.

Response

[UndeleteResult](#)

Faults

[UnexpectedErrorFault](#)

SEE ALSO:

[delete\(\)](#)

UndeleteResult

The `undelete()` call returns an `undeleteResult` object with the following properties:

Name	Type	Description
<code>id</code>	<code>ID</code>	ID of the record being undeleted.
<code>success</code>	<code>boolean</code>	Indicates whether the undelete was successful (<code>true</code>) or not (<code>false</code>).
<code>errors</code>	<code>Error[]</code>	If an error occurred during the <code>undelete()</code> call, an array of one or more <code>Error</code> objects providing the error code and description.

`update()`

Updates one or more existing records in your organization's data.

Syntax

```
SaveResult[] = connection.update(sObject[] sObjects);
```

Usage

Use this call to update one or more existing records, such as accounts or contacts, in your organization's data. The `update()` call is analogous to the `UPDATE` statement in SQL.

Permissions

Your client application must be logged in with sufficient access rights to `update()` records objects for the specified object, as well as individual fields inside that object. For more information, see [Factors that Affect Data Access](#).

Special Handling

Certain objects—and certain fields within those objects—require special handling or permissions. For example, you might also need permissions to access an object's parent object. Before you attempt to update a record for a particular object, be sure to read its description in the [Standard Objects](#) and in Salesforce Help.

Updateable Objects

Certain records cannot be updated via the API. To update a record via the `update()` call, its object must be configured as updateable (`updateable` is `true`). To determine whether an object can be updated, your client application can invoke the `describeSObjects()` call on the object and inspect its `updateable` property.

Required Fields

When updating required fields, you must supply a value—you cannot set the value to `null`. For more information, see [Required Fields](#).

ID Fields

Fields whose names contain "Id" are either that object's primary key (see [ID Field Type](#)) or a foreign key (see [Reference Field Type](#)). Client applications cannot update primary keys, but they can update foreign keys. For example, a client application can update the `OwnerId` of an [Account](#), because `OwnerId` is a foreign key that refers to the user who owns the account record. Use `describeSObjects()` to confirm whether the field can be updated.

This call checks a batch for duplicate `Id` values, and if there are duplicates, the first 12 are processed. For additional duplicate `Id` values, the [SaveResult](#) for those entries is marked with an error similar to the following:

```
Maximum number of duplicate updates in one batch (12 allowed).
```

Automatically Updated Fields

The API updates certain fields automatically, such as `LastModifiedDate`, `LastModifiedById`, and `SystemModstamp`. You cannot explicitly specify these values in your `update()` call.

Resetting Values to null

To reset a field value to `null`, you add the field name to the `fieldsToNull` array in the `sObject`. You cannot set required fields (`nillable` is `false`) to `null`.

Valid Field Values

You must supply values that are valid for the field's data type, such as integers (not alphabetic characters) for integer fields. In your client application, follow the data formatting rules specified for your programming language and development tool (your development tool will handle the appropriate mapping of data types in SOAP messages).

String Values

When storing values in string fields, the API trims any leading and trailing white space. For example, if the value of a name field is entered as " ABC Company ", then the value is stored in the database as "ABC Company".

Starting with API version 15.0, if you specify a value for a field that contains a string, and the value is too big for the field, the call fails and an error is returned. In previous versions of the API the value was truncated and the call succeeded. If you wish to keep the old behavior with versions 15.0 and later, use the [AllowFieldTruncationHeader](#) SOAP header.

Assignment Rules

When updating Case or Lead objects, your client application can set [AssignmentRuleHeader](#) options to have the case or lead automatically assigned to one or more users based on assignment rules configured in the Salesforce user interface. For more information, see [Case](#) or [Lead](#).

Maximum Number of Objects Updated

Your client application can change up to 200 records in a single `update()` call. If an update request exceeds 200 records, the entire operation fails.

Rollback on Error

The [AllOrNoneHeader](#) header allows you to roll back all changes unless all records are processed successfully. This header is available in API version 20.0 and later. Allows a call to roll back all changes unless all records are processed successfully.

Automatic Subscriptions for Chatter Feeds

To subscribe to records they create, users must enable the `Automatically follow records that I create` option in their personal settings. If users have automatic subscriptions enabled, they automatically follow the records they create and see changes to those records in their Chatter feed on the Home tab.

When you update the owner of a record, the new owner is not automatically subscribed to the record, unless the new owner has automatic subscriptions for records enabled in his or her Chatter feed settings. The previous owner is not automatically unsubscribed. If the new owner has automatic subscriptions for records enabled, the new and previous owners both see any changes to the record in their news feed.

A user can subscribe to a record or to another user. Changes to the record and updates from the users are displayed in the Chatter feed on the user's home page, which is a useful way to stay up-to-date with other users and with changes made to records in Salesforce. Feeds are available in API version 18.0 and later.

Updating Records for Different Object Types

You can update records for multiple object types, including custom objects, in one call with API version 20.0 and later. For example, you could update a contact and an account in one call. You can update records for up to 10 objects types in one call.

Records are saved in the same order that they are entered in the `sObjects` input array.


Records for different object types are broken into multiple chunks by Salesforce. A chunk is a subset of the `sObjects` input array and each chunk contains records of one object type. Data is committed on a chunk-by-chunk basis. Any Apex triggers related to the records in a chunk are invoked once per chunk. Consider an `sObjects` input array containing the following set of records:

```
account1, account2, contact1, contact2, contact3, case1, account3, account4, contact4
```

Salesforce splits the records into five chunks:

1. `account1, account2`
2. `contact1, contact2, contact3`
3. `case1`
4. `account3, account4`
5. `contact4`

Each call can process up to 10 chunks. If the `sObjects` array contains more than 10 chunks, you must process the records in more than one call.

 **Warning:** You can't update records for multiple object types in one call if one of those types is related to a feature in the Setup area in Salesforce. The only exceptions are the following objects:

- Custom settings objects, which are similar to custom objects. For more information, see “Create Custom Settings” in Salesforce Help.
- `GroupMember`
- `Group`
- `User` if the following fields are not being updated:
 - `UserRoleId`
 - `IsActive`
 - `ForecastEnabled`
 - `IsPortalEnabled`
 - `Username`
 - `ProfileId`

update () and Foreign Keys

You can use external ID fields as a foreign key, which allows you to update a record and relate it to another existing record in a single step instead of querying the parent record ID first. To do this, set the foreign key to an instance of the parent `sObject` that has only the external ID field specified. This external ID should match the external ID value on the parent record.

The following Java and C# examples show you how to update an opportunity and relate it to an existing account using a custom external ID field named `MyExtId__c`. Each example has a method that accepts the ID of the opportunity to update. It creates an opportunity `sObject` and sets its ID field so that the object points to an existing opportunity to be updated, sets a new value for the stage name field, and then sets the external ID field to the account object. It then updates the opportunity. Once the opportunity is updated, the account becomes its parent and the state name is updated.

Java Example

```
public void updateForeignKeySample(String oppId) {
    try {
        Opportunity updateOpportunity = new Opportunity();
        // Point to an existing opportunity to update
    }
}
```

```

updateOpportunity.setId(oppId);
updateOpportunity.setStageName("Qualification");

Account parentAccountRef = new Account();
parentAccountRef.setMyExtId__c("SAP1111111");
updateOpportunity.setAccount(parentAccountRef);

SaveResult[] results = connection
    .update(new SObject[] { updateOpportunity });
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

C# Example

```

public void updateForeignKeySample(String oppId)
{
    try
    {
        Opportunity updateOpportunity = new Opportunity();
        // Point to an existing opportunity to update
        updateOpportunity.Id = oppId;
        updateOpportunity.StageName = "Prospecting";

        Account parentAccountRef = new Account();
        parentAccountRef.MyExtId__c = "SAP1111111";
        updateOpportunity.Account = parentAccountRef;

        SaveResult[] results = binding.update(
            new sObject[] { updateOpportunity });
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}

```

Basic Steps for Updating Records

Use this process to update records:

1. Determine the ID of each record that you want to `update()`. For example, you might call `query()` to retrieve a set of records (with their IDs), based on specific criteria, that you would want to update. If you know the ID of the record that you want to update, you can call `retrieve()` instead. For information on IDs, see [ID Field Type](#).
2. Create an `sObject` for each record, and populate its fields with the data that you want to update.
3. Construct an `sObject[]` array and populate that array with the records that you want to update.
4. Call `update()`, passing in the `sObject[]` array.
5. Process the results in the `SaveResult[]` object to verify whether the records have been successfully updated.

Sample Code—Java

This sample accepts the IDs of the accounts to update. It creates two account sObjects, sets each with one of the passed IDs so that the sObject points to an existing account, and sets other fields. It then makes the update() call and verifies the results.

```
public void updateRecords(String[] ids) {
    Account[] updates = new Account[2];

    Account account1 = new Account();
    account1.setId(ids[0]);
    account1.setShippingPostalCode("89044");
    updates[0] = account1;

    Account account2 = new Account();
    account2.setId(ids[1]);
    account2.setNumberOfEmployees(1000);
    updates[1] = account2;

    // Invoke the update call and save the results
    try {
        SaveResult[] saveResults = connection.update(updates);
        for (SaveResult saveResult : saveResults) {
            if (saveResult.isSuccess()) {
                System.out.println("Successfully updated Account ID: "
                    + saveResult.getId());
            } else {
                // Handle the errors.
                // We just print the first error out for sample purposes.
                Error[] errors = saveResult.getErrors();
                if (errors.length > 0) {
                    System.out.println("Error: could not update " + "Account ID "
                        + saveResult.getId() + ".");
                    System.out.println("\tThe error reported was: ("
                        + errors[0].getStatusCode() + ") "
                        + errors[0].getMessage() + ".");
                }
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample accepts the IDs of the accounts to update. It creates two account sObjects, sets each with one of the passed IDs so that the sObject points to an existing account, and sets other fields. It then makes the update() call and verifies the results.

```
public void updateRecords(String[] ids)
{
    Account[] updates = new Account[2];

    Account account1 = new Account();
    account1.Id = ids[0];
```

```

account1.ShippingPostalCode = "89044";
updates[0] = account1;

Account account2 = new Account();
account2.Id = ids[1];
account2.NumberOfEmployees = 1000;
updates[1] = account2;

// Invoke the update call and save the results
try
{
    SaveResult[] saveResults = binding.update(updates);
    foreach (SaveResult saveResult in saveResults)
    {
        if (saveResult.success)
        {
            Console.WriteLine("Successfully updated Account ID: " +
                saveResult.id);
        }
        else
        {
            // Handle the errors.
            // We just print the first error out for sample purposes.
            Error[] errors = saveResult.errors;
            if (errors.Length > 0)
            {
                Console.WriteLine("Error: could not update " +
                    "Account ID " + saveResult.id + ".");
            };
            Console.WriteLine("\tThe error reported was: (" +
                errors[0].statusCode + ") " +
                errors[0].message + ".");
        };
    }
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

Arguments

Name	Type	Description
sObjects	sObject[]	Array of one or more records (maximum of 200) to update.

Response

[SaveResult\[\]](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[API Call Basics](#)

SaveResult


The `update()` call returns an array of `SaveResult` objects. Each element in the `SaveResult` array corresponds to the `sObject[]` array passed as the `sObjects` parameter in the `update()` call. For example, the object returned in the first index in the `SaveResult` array matches the object specified in the first index of the `sObject[]` array.

A `SaveResult` object has the following properties:

Name	Type	Description
<code>id</code>	ID	ID of an <code>sObject</code> that you successfully updated. If this field contains a value, then the object was updated successfully. If this field is empty, then the object was not updated and the API returned error information instead.
<code>success</code>	boolean	Indicates whether the <code>update()</code> call succeeded (<code>true</code>) or not (<code>false</code>) for this object.
<code>errors</code>	<code>Error[]</code>	If an error occurred during the <code>update()</code> call, an array of one or more <code>Error</code> objects providing the error code and description. If your organization has active duplicate rules and a duplicate is detected, the <code>SaveResult</code> includes an <code>Error</code> with a data type of <code>DuplicateError</code> .

upsert()

Creates new records and updates existing records; uses a custom field to determine the presence of existing records. In most cases, we recommend that you use `upsert()` instead of `create()` to avoid creating unwanted duplicate records (idempotent). Available in the API version 7.0 and later.

 **Note:** Starting with API version 15.0, if you specify a value for a field that contains a string, and the value is too big for the field, the call fails and an error is returned. In previous versions of the API the value was truncated and the call succeeded. If you wish to keep the old behavior with versions 15.0 and later, use the `AllowFieldTruncationHeader` SOAP header.

Syntax

```
UpsertResult[] = connection.upsert(String externalIdFieldName, sObject[] sObjects);
```


Usage

Upsert is a merging of the words insert and update. This call is available for objects if the object has an external ID field or a field with the `idLookup` field property.

On custom objects, this call uses an indexed custom field called an external ID to determine whether to create a new record or update an existing record. On standard objects, this call can use the name of any field with the `idLookup` instead of the external ID.


 **Note:** External ID fields cannot be used with `merge()`.

For more information about adding custom fields, including external ID fields, to objects, see the “Adding Fields” topic in Salesforce Help.

Using this call can dramatically reduce how many calls you need to make, particularly when:

- You are integrating your organization’s Salesforce data with ERP (enterprise resource planning) systems such as accounting and manufacturing.
- You are importing data and want to prevent the creation of duplicate objects.

If you are upserting a record for an object that has a custom field with both the `EXTERNAL ID` and `UNIQUE` attributes selected (a unique index), you do not need any special permissions, because the `UNIQUE` attribute prevents the creation of duplicates. If you are upserting a record for an object that has the `EXTERNAL ID` attribute selected but not the `UNIQUE` attribute selected, (a non-unique index) your client application must have the permission “View All Data” to execute this call.

 **Note:** Matching by external ID is case-insensitive only if the external ID field has the `UNIQUE` attribute and the `TREAT "ABC" AND "abc" AS DUPLICATE VALUES (CASE INSENSITIVE)` option selected. These options are selected in the Salesforce user interface during field creation. If this is the case, “ABC123” is matched with “abc123.” Before performing an operation, if you have external ID fields without the case-insensitive option selected, review your external IDs for any values that would be matched if case was not considered. If such values exist, you may want to modify them to make them unique, or select the case-sensitive option for your external ID fields. For more information about field attributes, see “Custom Field Attributes” in Salesforce Help.

How Upsert Chooses to `update()` Or `create()`

Upsert uses the external ID to determine whether it should create a new record or update an existing one:

- If the external ID is not matched, then a new record is created.
- If the external ID is matched once, then the existing record is updated.
- If the external ID is matched multiple times, then an error is reported.
- When batch updating multiple records where the external ID is the same for two or more records in your batch call, those records will be marked as errors in the `UpsertResult` file. The records will be neither created or updated.

Rollback on Error

The `AllOrNoneHeader` header allows you to roll back all changes unless all records are processed successfully. This header is available in API version 20.0 and later. Allows a call to roll back all changes unless all records are processed successfully.

Automatic Subscriptions for Chatter Feeds

To subscribe to records they create, users must enable the `Automatically follow records that I create` option in their personal settings. If users have automatic subscriptions enabled, they automatically follow the records they create and see changes to those records in their Chatter feed on the Home tab.

When you update the owner of a record, the new owner is not automatically subscribed to the record, unless the new owner has automatic subscriptions for records enabled in his or her Chatter feed settings. The previous owner is not automatically unsubscribed. If the new owner has automatic subscriptions for records enabled, the new and previous owners both see any changes to the record in their news feed.

A user can subscribe to a record or to another user. Changes to the record and updates from the users are displayed in the Chatter feed on the user's home page, which is a useful way to stay up-to-date with other users and with changes made to records in Salesforce. Feeds are available in API version 18.0 and later.

upsert () and Foreign Keys

You can use external ID fields as a foreign key, which allows you to create or update a record and relate it to another existing record in a single step instead of querying the parent record ID first. To do this, set the foreign key to an instance of the parent sObject that has only the external ID field specified. This external ID should match the external ID value on the parent record. Unlike `create ()`, the parent record must already exist when using `upsert ()` to create or update a child record related by a foreign key.

The following Java and C# examples upsert an opportunity. In this case, the opportunity doesn't exist in the database, so the `upsert ()` call will create it. The opportunity references an existing account. Rather than specify the account ID, which would require a separate query to obtain, we specify an external ID for the account, in this example the `MyExtId__c` custom field.

Java Example

```
public void upsertForeignKeySample() {
    try {
        Opportunity newOpportunity = new Opportunity();
        newOpportunity.setName("UpsertOpportunity");
        newOpportunity.setStageName("Prospecting");
        Calendar dt = connection.getServerTimestamp().getTimestamp();
        dt.add(Calendar.DAY_OF_MONTH, 7);
        newOpportunity.setCloseDate(dt);
        newOpportunity.setMyExtId__c("UPSERTID001");

        // Parent Account record must already exist
        Account parentAccountRef = new Account();
        parentAccountRef.setMyExtId__c("SAP111111");
        newOpportunity.setAccount(parentAccountRef);

        SaveResult[] results = connection
            .upsert("MyExtId__c", new SObject[] { newOpportunity });
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

C# Example

```
public void upsertForeignKeySample()
{
    try
    {
        Opportunity newOpportunity = new Opportunity();
        newOpportunity.Name = "UpsertOpportunity";
        newOpportunity.StageName = "Prospecting";
        DateTime dt = (DateTime)binding.getServerTimestamp().timestamp;
        newOpportunity.CloseDate = dt.AddDays(7);
    }
}
```

```

newOpportunity.CloseDateSpecified = true;
newOpportunity.MyExtId__c = "UPSERTID001";

// Parent Account record must already exist
Account parentAccountRef = new Account();
parentAccountRef.MyExtId__c = "SAP111111";
newOpportunity.Account = parentAccountRef;

SaveResult[] results = binding
    .upsert("MyExtId", new sObject[] { newOpportunity });
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

Sample Code—Java

This sample upserts two accounts using a custom external ID field called `MyExtId__c`. The `upsert()` call matches the accounts based on the `MyExtId__c` field in order to determine whether to create or update the accounts. Before running this sample, change the `MyExtId__c` field name to an existing custom ID field name in your org.

```

public void upsertRecords() {
    SObject[] upserts = new Account[2];

    Account upsertAccount1 = new Account();
    upsertAccount1.setName("Begonia");
    upsertAccount1.setIndustry("Education");
    upsertAccount1.setMyExtId__c("1111111111");
    upserts[0] = upsertAccount1;

    Account upsertAccount2 = new Account();
    upsertAccount2 = new Account();
    upsertAccount2.setName("Bluebell");
    upsertAccount2.setIndustry("Technology");
    upsertAccount2.setMyExtId__c("2222222222");
    upserts[1] = upsertAccount2;

    try {
        // Invoke the upsert call and save the results.
        // Use External_Id custom field for matching records.
        UpsertResult[] upsertResults = connection.upsert(
            "MyExtId__c", upserts);
        for (UpsertResult result : upsertResults) {
            if (result.isSuccess()) {
                System.out.println("\nUpsert succeeded.");
                System.out.println((result.isCreated() ? "Insert" : "Update")
                    + " was performed.");
                System.out.println("Account ID: " + result.getId());
            } else {
                System.out.println("The Upsert failed because: "

```

```

        + result.getErrors()[0].getMessage());
    }
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

Sample Code—C#

This sample upserts two accounts using a custom external ID field called `MyExtId__c`. The `upsert()` call matches the accounts based on the `MyExtId__c` field in order to determine whether to create or update the accounts. Before running this sample, change the `MyExtId__c` field name to an existing custom ID field name in your org.

```

public void upsertRecords()
{
    sObject[] upserts = new Account[2];

    Account upsertAccount1 = new Account();
    upsertAccount1.Name = "Begonia";
    upsertAccount1.Industry = "Education";
    upsertAccount1.MyExtId__c = "1111111111";
    upserts[0] = upsertAccount1;

    Account upsertAccount2 = new Account();
    upsertAccount2 = new Account();
    upsertAccount2.Name = "Bluebell";
    upsertAccount2.Industry = "Technology";
    upsertAccount2.MyExtId__c = "2222222222";
    upserts[1] = upsertAccount2;

    try
    {
        // Invoke the upsert call and save the results.
        // Use External_Id custom field for matching records.
        UpsertResult[] upsertResults =
            binding.upsert("MyExtId__c", upserts);
        foreach (UpsertResult result in upsertResults)
        {
            if (result.success)
            {
                Console.WriteLine("\nUpsert succeeded.");
                Console.WriteLine(
                    (result.created ? "Insert" : "Update") +
                    " was performed.");
            };
            Console.WriteLine("Account ID: " + result.id);
        }
        else
        {
            Console.WriteLine("The Upsert failed because: " +
                result.errors[0].message);
        }
    }
}

```

```

    }
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}

```

Arguments

Name	Type	Description
ExternalIDFieldName	string	Contains the name of the field on this object with the external ID field attribute for custom objects or the idLookup field property for standard objects. The idLookup field property is usually on a field that is the object's ID field or name field, but there are exceptions, so check for the presence of the property in the object you wish to <code>upsert()</code> .
sObjects	sObject[]	Array of one or more records (maximum of 200) to create or update. All records must have the same object type.

Response

[UpsertResult\[\]](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[create\(\)](#)

[update\(\)](#)

[API Call Basics](#)

UpsertResult

The `upsert` call returns an array of `UpsertResult` objects. Each element in the array corresponds to the `sObject[]` array passed as the `sObjects` parameter in the `upsert()` call. For example, the object returned in the first index in the `UpsertResult` array matches the object specified in the first index of the `sObject[]` array.


An `UpsertResult` object has the following properties:

Name	Type	Description
created	boolean	Indicates whether the record was created (<code>true</code>) or updated (<code>false</code>).

Name	Type	Description
errors	Error[]	If errors occurred during the call, an array Error objects, providing the error code and description, is returned.
id	ID	If the call succeeded, the field contains the ID of the record that was either updated or created. If there was an error, the field is null. For more information, see ID Field Type .
success	boolean	Indicates whether the call succeeded (<code>true</code>) or not (<code>false</code>) for this object. If your organization has active duplicate rules and a duplicate is detected, the UpsertResult includes an Error with a data type of DuplicateError .

CHAPTER 10 Describe Calls

The following table lists supported describe calls in the API in alphabetical order, and provides a brief description for each. Click a call name to see syntax, usage, and more information for that call.

 **Note:** For a list of Apex-related calls, see [Apex-Related Calls](#), for a list of core calls, see [Core Calls](#), and for a list of utility calls, see [Utility Calls](#).

Call	Description
describeAllTabs ()	Returns information about all the tabs—including Lightning page tabs—available to the logged-in user, regardless of whether the user has chosen to hide tabs in his own user interface via the All Tabs (+) tab customization feature.
describeAppMenu ()	Retrieves metadata about items either in the Salesforce mobile app navigation menu or the Salesforce drop-down app menu.
describeApprovalLayout ()	Retrieves metadata about approval layouts for the specified object type.
describeAvailableQuickActions ()	In API version 28.0, describes details about actions available for a specified parent. In API version 29.0 and greater, describes details about actions available for a specified context.
describeCompactLayouts ()	Retrieves metadata about compact layouts for the specified object type.
describeDataCategoryGroups ()	Retrieves available category groups for entities specified in the request.
describeDataCategoryGroupStructures ()	Retrieves available category groups along with their data category structure for entities specified in the request.
describeGlobal ()	Retrieves a list of available objects for your organization's data.
describeGlobalTheme ()	Returns information about both objects and themes available to the current logged-in user.
describeKnowledge ()	Retrieves the Knowledge language settings in the organization.
describeLayout ()	Retrieves metadata about page layouts for the specified object type.
describePrimaryCompactLayouts ()	Retrieves metadata about the primary compact layout for each of the specified object types.
describeQuickActions ()	Retrieves details about specified actions.
describeSearchScopeOrder ()	Retrieves an ordered list of objects in the logged-in user's default global search scope, including any pinned objects in the user's search results page.
describeSObject ()	Retrieves metadata (field list and object properties) for the specified object type. Superseded by describeSObjects () .
describeSObjects ()	An array-based version of describeSObject .
describeSoftphoneLayout ()	Describes the softPhone layout(s) created for an organization.

Call	Description
<code>describeSoqlListViews()</code>	Retrieves the SOQL query and other information about a list view.
<code>describeTabs()</code>	Returns information about the standard and custom apps available to the logged-in user, as listed in the Lightning Platform app menu at the top of the page.
<code>describeTheme()</code>	Returns information about themes available to the current logged-in user.

Samples

The samples in this section are based on the enterprise WSDL file. They assume that you have already imported the WSDL file and created a connection. To learn how to do so, see the [Quick Start](#) tutorial.

describeAllTabs()

Returns information about all the tabs—including Lightning page tabs—available to the logged-in user, regardless of whether the user has chosen to hide tabs in his own user interface via the All Tabs (+) tab customization feature.

Syntax

```
DescribeTab [] = connection.describeAllTabs();
```

Usage

Use the `describeAllTabs()` call to obtain information about all the tabs that are available to the logged-in user.

Alternately, use `describeTabs()` if you want information only about the tabs that display in the Salesforce user interface for the logged-in user.

Sample Code—Java

This sample calls `describeAllTabs()`, which returns an array of `DescribeTab` results.

```
public void describeAllTabsSample() {
    try {
        // Describe tabs
        DescribeTab[] tabs = connection.describeAllTabs();
        System.out.println("There are " + tabs.length +
            " tabs available to you.");

        // Iterate through the returned tabs
        for (int j = 0; j < tabs.length; j++) {
            DescribeTab tab = tabs[j];
            System.out.println("\tTab " + (j + 1) + ":");
            System.out.println("\t\tName: " + tab.getName());
            System.out.println("\t\tAssociated SObject" + tab.getObjectName());
            System.out.println("\t\tLabel: " + tab.getLabel());
        }
    }
}
```



```

        System.out.println("\t\tURL: " + tab.getUrl());
        DescribeColor[] tabColors = tab.getColors();
        // Iterate through tab colors as needed
        DescribeIcon[] tabIcons = tab.getIcons();
        // Iterate through tab icons as needed
    }
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

Arguments

None.

Response

[DescribeTab](#)

describeAppMenu ()

Retrieves metadata about items either in the Salesforce mobile app navigation menu or the Salesforce drop-down app menu. This call is available in API version 29.0 and later.

If you're accessing the API using a custom community URL, the `describeAppMenu ()` call retrieves the tab set associated with the community ID you specify.

Syntax

```
DescribeAppMenuResult describeResult = connection.describeAppMenu(String appMenuType,
String networkId);
```

Code Sample—Java

This code sample shows how to get the menu items from the Salesforce mobile app navigation menu.

```

public void describeAppMenu() {
    try {
        //The following two lines are equivalent
        DescribeAppMenuResult describe = connection.describeAppMenu("Salesforce1", "");
        DescribeAppMenuResult appMenu = getClient().describeAppMenu(AppMenuType.Salesforce1);

        for (DescribeAppMenuItem menuItem : appMenu.getAppMenuItems()) {

            if (menuItem.getType() == "Tab.apexPage") {

                String visualforceUrl = menuItem.getContent();
            }
        }
    }
}

```

```

        System.out.println("URL to Visualforce page: " + visualforceUrl);
    }
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

Arguments

Name	Type	Description
appMenuType	string	Restricts the menu data returned to the specified menu type. Valid values are: <ul style="list-style-type: none"> AppSwitcher—to retrieve the data from the Salesforce drop-down app menu Salesforce1—to retrieve the data from the Salesforce mobile app navigation menu NetworkTabs—to retrieve the data from a community tab set
networkId	ID	If the appMenuType is set to NetworkTabs, enter the ID of the community to retrieve the tab set from. If appMenuType is not NetworkTabs, this field must be null or empty.

Response

[DescribeAppMenuResult](#)

Faults

[InvalidOrNullForRestrictedPicklist](#)

DescribeAppMenuResult

The `describeAppMenu()` call returns a list of menu items contained in the specified menu type. The following types are available in API version 29.0 and later.

Name	Type	Description
appMenuItems	DescribeAppMenuItem[]	Array of one or more menu items in the selected menu type.

DescribeAppMenuItem

Each `DescribeAppMenuItem` object has these fields:

Name	Type	Description
colors	DescribeColor[]	Array of color information used for the tab associated with the menu item.
content	string	Information that helps build the menu item. Each menu item has a different type of content for this field. For example, the Salesforce app menu type could contain: <ul style="list-style-type: none"> FlexiPage—the ID of the Lightning page Visualforce tab—the URL to the page, such as <code>/apex/myApexPage</code>. Menu items of types other than these don't use this field.
icons	DescribeColor[]	Array of icon information used for the tab associated with the menu item.
label	string	The display label of the menu item.
name	string	API name of the menu item.
type	string	The type of menu item, and its subtype, if any. Possible values for the Salesforce app menu type are: <ul style="list-style-type: none"> Standard.Dashboards—Dashboards menu item Standard.Feed—Chatter feed menu item Standard.Today—the Today menu item Standard.Tasks—Tasks menu item Tab.apexPage—a Visualforce tab menu item Tab.flexipage—a Lightning page tab menu item
url	string	The Salesforce URL the menu item should point to. For the Salesforce app menu type, this field is <code>null</code> for the Dashboards, Feed, Today, Tasks, and Lightning page menu items.

describeApprovalLayout()

Retrieves metadata about approval layouts for the specified object type.

Syntax

```
DescribeApprovalLayoutResult approvalLayoutResult = connection.describeApprovalLayout(string sObjectType, string[] approvalProcessNames);
```

Usage

Use this call to retrieve information about the approval layout for a given object type. Each approval process has one approval layout.

If you supply a null value for `approvalProcessNames`, all the approval layouts for the object are returned, instead of the approval layout of each specified approval process.

Sample Code—Java

This sample shows how to get the approval layouts of an `Account` sObject. It calls `describeApprovalLayout()` with the name of the sObject type to describe. After getting the approval layouts, the sample prints the name and fields found for each approval layout.

```
public void describeApprovalLayoutSample() {
    try {
        String objectToDescribe = "Account";
        DescribeApprovalLayoutResult approvalLayoutResult =
            connection.describeApprovalLayout(objectToDescribe, null);
        System.out.print("There are " + approvalLayoutResult.getApprovalLayouts().length);
        System.out.println(" approval layouts for the " + objectToDescribe + " object.");

        // Get all the approval layouts for the sObject
        for (int i = 0; i < approvalLayoutResult.getApprovalLayouts().length; i++) {
            DescribeApprovalLayout aLayout = approvalLayoutResult.getApprovalLayouts()[i];
            System.out.println(" There is an approval layout with name: " + aLayout.getName());

            DescribeLayoutItem[] layoutItems = aLayout.getLayoutItems();
            System.out.print(" There are " + layoutItems.length);
            System.out.println(" fields in this approval layout.");
            for (int j = 0; j < layoutItems.length; j++) {
                System.out.print("This approval layout has a field with name: ");
                System.out.println(layoutItems[j].getLabel());
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Arguments

Name	Type	Description
<code>sObjectType</code>	string	The specified value must be a valid object for your organization. If the object is a person account, specify <code>Account</code> , or if it is a person contact, specify <code>Contact</code> .
<code>approvalProcessNames</code>	string[]	Optional array of the approval process API names to return approval layout metadata for.

Response

[DescribeApprovalLayoutResult](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

DescribeApprovalLayoutResult

The `describeApprovalLayout ()` call returns a `DescribeApprovalLayoutResult` object containing top-level record type information about the passed-in `sObjectType`. Your client application can traverse this object to retrieve detailed metadata about the approval layout.

Name	Type	Description
<code>approvalLayouts</code>	DescribeApprovalLayout[]	List of all the approval layouts in use by the object.

DescribeApprovalLayout

Represents an individual item in the [DescribeApprovalLayout](#) list.

Name	Type	Description
<code>id</code>	ID	Unique ID of this ApprovalLayout. For information on IDs, see ID Field Type .
<code>label</code>	string	Label of the approval layout.
<code>layoutItems</code>	DescribeLayoutItem[]	Array of one or more fields assigned to the approval layout.
<code>name</code>	string	API name of the approval layout.

describeAvailableQuickActions ()

In API version 28.0, describes details about actions available for a specified parent. In API version 29.0 and greater, describes details about actions available for a specified context.

Syntax

```
DescribeAvailableQuickActionResult [] = connection.describeAvailableQuickActions(string
parentOrContextType );
```

Usage

Use `describeAvailableQuickActions ()` to get the list of actions whose parent (API version 28.0) or context (API version 29.0 and greater) entity name is supplied as well as standard and global actions. The `describeAvailableQuickActions ()` call uses the parent entity name, such as "Account", or "null" for global actions, or in API version 29.0 and greater, the context, to return an array of `DescribeAvailableQuickActionResult`.

Sample—Java

This sample retrieves and displays the available action information for the Account object.

```
public void example() throws Exception {
    DescribeAvailableQuickActionResult[] aResult =
        conn.describeAvailableQuickActions("Account");
    for(DescribeAvailableQuickActionResult ar : aResult) {
        System.out.println("Action label: " + ar.getLabel());
        System.out.println("Action name: " + ar.getName());
        System.out.println("Action type: " + ar.getType());
    }
}
```

Arguments

Name	Type	Description
parentOrContextType	string	Either a standard or custom object. <ul style="list-style-type: none"> The <code>parentType</code> applies only to API version 28.0. The <code>contextType</code> applies to API version 29.0 and greater.

Response

An array of [DescribeAvailableQuickActionResult](#) objects.

Faults

`connection.exception` errors

DescribeAvailableQuickActionResult

The `describeAvailableQuickActions()` call returns an array of `DescribeAvailableQuickActionResult` objects. In API version 28.0, each `DescribeAvailableQuickActionResult` object represents details about actions available for a specified parent. In API version 29.0 and greater, each `DescribeAvailableQuickActionResult` object represents details about actions available for a specified context.

Name	Type	Description
actionEnumOrId	string	The unique ID for the action. If the action doesn't have an ID, its API name is used. This field is available in API version 35.0 and later.
label	string	The action label.
name	string	The action name.
type	string	<ul style="list-style-type: none"> LogACall

Name	Type	Description
		<ul style="list-style-type: none"> • SocialPost • Canvas • Create • VisualforcePage • Update

describeCompactLayouts ()

Retrieves metadata about compact layouts for the specified object type.

Syntax

```
DescribeCompactLayoutsResult compactLayoutResult = connection.describeCompactLayouts(string
  sObjectType, ID[] recordTypeId);
```

Usage

Use this call to retrieve information about the compact layout for a given object type. This call returns metadata about a given compact layout, including the record type mappings. For more information about compact layouts, see the Salesforce online help.

Sample Code—Java

This sample shows how to get the compact layouts of an Account sObject. It calls `describeCompactLayouts ()` with the name of the sObject type to describe. After getting the compact layouts, the sample prints the images, fields, and action buttons found for each compact layout. Next, it prints the system default compact layout for the object, then the mapping information of record types to compact layouts.

```
public void testDescribeCompactLayoutsSample() {
    try {
        String objectToDescribe = "Account";
        DescribeCompactLayoutsResult compactLayoutResult = connection
            .describeCompactLayouts(objectToDescribe, null);
        System.out.println("There are " + compactLayoutResult.getCompactLayouts().length
            + " compact layouts for the " + objectToDescribe + " object.");

        // Get all the compact layouts for the sObject
        for (int i = 0; i < compactLayoutResult.getCompactLayouts().length; i++) {
            DescribeCompactLayout cLayout = compactLayoutResult.getCompactLayouts()[i];
            System.out.println(" There is a compact layout with name: " + cLayout.getName());

            DescribeLayoutItem[] fieldItems = cLayout.getFieldItems();
            System.out.println(" There are " + fieldItems.length + " fields in this compact
                layout.");

            // Write field items
```

```

        for (int j = 0; j < fieldItems.length; j++) {
            System.out.println(j + " This compact layout has a field with name: " +
fieldItems[j].getLabel());
        }

        DescribeLayoutItem[] imageItems = cLayout.getImageItems();
        System.out.println(" There are " + imageItems.length + " image fields in this
compact layout.");

        // Write the image items
        for (int j = 0; j < imageItems.length; j++) {
            System.out.println(j + " This compact layout has an image field with name:
" + imageItems[j].getLabel());
        }

        DescribeLayoutButton[] actions = cLayout.getActions();
        System.out.println(" There are " + actions.length + " buttons in this compact
layout.");

        // Write the action buttons
        for (int j = 0; j < actions.length; j++) {
            System.out.println(j + " This compact layout has a button with name: " +
actions[j].getLabel());
        }

        System.out.println("This object's default compact layout is: "
+ compactLayoutResult.getDefaultCompactLayoutId());

        RecordTypeCompactLayoutMapping[] mappings =
compactLayoutResult.getRecordTypeCompactLayoutMappings();
        System.out.println("There are " + mappings.length + " record type to compact
layout mapping for the "
+ objectToDescribe + " object.");
        for (int j = 0; j < mappings.length; j++) {
            System.out.println(j + " Record type " + mappings[j].getRecordTypeId()
+ " is mapped to compact layout " +
mappings[j].getCompactLayoutId());
        }
    }

} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

Arguments

Name	Type	Description
sObjectType	string	The specified value must be a valid object for your organization. If the object is a person account, specify Account, or if it is a person contact, specify Contact.

Name	Type	Description
<code>recordTypeId</code>	ID[]	Optional parameter that restricts the compact layout data returned to the specified record types.

Response

[DescribeCompactLayoutsResult](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

DescribeCompactLayoutsResult

The `describeCompactLayouts()` call returns a `DescribeCompactLayoutsResult` object containing top-level record type information about the passed-in `sObjectType`, as well as a mapping of record types to compact layouts. Your client application can traverse this object to retrieve detailed metadata about the compact layout.

Name	Type	Description
<code>compactLayouts</code>	DescribeCompactLayout []	List of all the compact layouts in use by the object.
<code>defaultCompactLayoutId</code>	ID	ID of the primary compact layout assigned to the object. The system default compact layout ID has a value of <code>null</code> .
<code>recordTypeCompactLayoutMappings</code>	RecordTypeCompactLayoutMapping []	Record type mapping(s) for the object. The compact layouts associated with the object may be mapped to more than one record type.

DescribeCompactLayout

Represents an individual item in the [DescribeCompactLayout](#) list.

Name	Type	Description
<code>actions</code>	DescribeLayoutButtonSection []	Array of one or more DescribeLayoutButtonSection items assigned to the compact layout. This list is set by Salesforce and is read-only.
<code>fieldItems</code>	DescribeLayoutItem []	Array of one or more fields assigned to the compact layout.
<code>id</code>	ID	Unique ID of this <code>CompactLayout</code> . For information on IDs, see ID Field Type .

Name	Type	Description
imageItems	DescribeLayoutItem[]	Array of one or more images assigned to the compact layout. This list is set by Salesforce and is read-only.
label	string	Label of the compact layout.
name	string	API name of the compact layout.
objectType	string	The name of the object to which the compact layout is assigned.

RecordTypeCompactLayoutMapping

Represents a single record type mapping in the `recordTypeCompactLayoutMappings` field in a `DescribeCompactLayoutsResult` object. This object is a map of valid `recordTypeId` to `compactLayoutId`.

Name	Type	Description
available	boolean	Indicates whether this record type is available (<code>true</code>) or not (<code>false</code>). Availability is used to display a list of available record types to the user when they are creating a new record.
compactLayoutId	ID	ID of the compact layout associated with this record type. This field has a value of <code>null</code> if the record type is associated with the system default compact layout.
compactLayoutName	string	API name of the compact layout.
recordTypeName	string	API name of the record type.
recordTypeId	ID	ID of the record type.

describeDataCategoryGroups ()

Retrieves available category groups for objects specified in the request.

Syntax

```
DescribeDataCategoryGroupResult[] = connection.describeDataCategoryGroups () (string[]
sObjectTypes) ;
```

Usage

Use this call to describe the available category groups for the objects specified in the request. This call can be used with the `describeDataCategoryGroupStructures ()` call to describe all the categories available for a specific object. For additional information about data categories, see “Work with Data Categories” in the Salesforce online help.

Sample Code—Java

This sample shows how to retrieve the data category groups associated with:

- Salesforce Knowledge articles
- Questions from the Answers feature

It returns the name, label and description of a category group and the name of the associated `subject` (article or question). It also returns the number of data categories in the data category group.

```
public void describeDataCategoryGroupsSample() {
    try {
        // Make the describe call for data category groups
        DescribeDataCategoryGroupResult[] results =
            connection.describeDataCategoryGroups(new String[] {
                "KnowledgeArticleVersion", "Question"});

        // Get the properties of each data category group
        for (int i = 0; i < results.length; i++) {
            System.out.println("sObject: " +
                results[i].getSubject());
            System.out.println("Group name: " +
                results[i].getName());
            System.out.println("Group label: " +
                results[i].getLabel());
            System.out.println("Group description: " +
                (results[i].getDescription()==null? "" :
                results[i].getDescription()));
            System.out.println("Number of categories: " +
                results[i].getCategoryCount());
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample shows how to retrieve the data category groups associated with:

- Salesforce Knowledge articles
- Questions from the Answers feature

It returns the name, label and description of a category group and the name of the associated `subject` (article or question). It also returns the number of data categories in the data category group.

```
public void describeDataCategoryGroups() {
    try {
        // Make the describe call for data category groups
        DescribeDataCategoryGroupResult[] results =
            binding.describeDataCategoryGroups(new String[] {
                "KnowledgeArticleVersion", "Question"});

        // Get the properties of each data category group
        for (int i = 0; i < results.Length; i++) {
```

```

Console.WriteLine("sObject: " +
results[i].subject);
Console.WriteLine("Group name: " +
results[i].name);
Console.WriteLine("Group label: " +
results[i].label);
Console.WriteLine("Group description: " +
(results[i].description==null? "" :
results[i].description));
Console.WriteLine("Number of categories: " +
results[i].categoryCount);
}
} catch (SoapException e) {
Console.WriteLine("An unexpected error has occurred: " +
e.Message + "\n" + e.StackTrace);
}
}

```

Arguments

Name	Type	Description
sObjectTypes	string[]	<p>The specified value can be:</p> <ul style="list-style-type: none"> KnowledgeArticleVersion—to retrieve category groups associated with article types. Question—to retrieve category groups associated with questions. <p>For additional information about articles and questions, see "Work with Articles and Translations" in the Salesforce online help.</p>

Response

DescribeDataCategoryGroupResult

Faults

InvalidSObjectFault

UnexpectedErrorFault

DescribeDataCategoryGroupResult

The describeDataCategoryGroups() call returns a DescribeDataCategoryGroupResult object containing the list of the category groups associated with the specified objects.

Name	Type	Description
categoryCount	int	The number of visible data categories in the data category group.

Name	Type	Description
description	string	The description of the data category group.
label	string	Label for the data category group in the Salesforce user interface.
name	string	The unique name used for API access to the data category group .
subject	string	The object associated with the data category group.

describeDataCategoryGroupStructures ()

Retrieves available category groups along with their data category structure for objects specified in the request.

Syntax

```
describeDataCategoryGroupStructures() [] = connection.
    describeDataCategoryGroupStructures() (DataCategoryGroupObjectTypePair []
    pairs, boolean topCategoriesOnly)
```

Usage

Use this call to return the visible data category structure for the given object category group pairs. First use [describeDataCategoryGroups \(\)](#) to find the available category groups for the objects specified. From the returned list, choose the object category group pairs to pass as the input in [describeDataCategoryGroupStructures \(\)](#). This call returns all the visible categories and data category structure as output. For additional information about data categories and data category visibility, see “Work with Data Categories” and “Data Category Visibility” in the Salesforce online help.

Sample Code—Java

This sample shows how to use sObject and data category group pairs to retrieve data categories for each pair. It calls [describeDataCategoryGroupStructures \(\)](#) with two pairs, KnowledgeArticleVersion/Regions and Question/Regions, and iterates through the results of this call. It gets the top categories for each result, which is “All”, and then gets the first-level child categories. The sample requires that you set up a data category group called *Regions* with some child categories and associate it with a knowledge article and questions. Alternatively, you can replace the data category group name in the sample if you want to use an existing data category group in your org that has a different name.

```
public void describeDataCategoryGroupStructuresSample () {
    try {
        // Create the data category pairs
        DataCategoryGroupObjectTypePair pair1 =
        new DataCategoryGroupObjectTypePair ();
        DataCategoryGroupObjectTypePair pair2 =
        new DataCategoryGroupObjectTypePair ();
        pair1.setSubject ("KnowledgeArticleVersion");
        pair1.setDataCategoryGroupName ("Regions");
        pair2.setSubject ("Question");
```

```
pair2.setDataCategoryGroupName("Regions");

DataCategoryGroupSubjectTypePair[] pairs =
new DataCategoryGroupSubjectTypePair[] {
pair1,
pair2
};

// Get the list of top level categories using the describe call
DescribeDataCategoryGroupStructureResult[] results =
connection.describeDataCategoryGroupStructures(
pairs,
false
);

// Iterate through each result and get some properties
// including top categories and child categories
for (int i = 0; i < results.length; i++) {
DescribeDataCategoryGroupStructureResult result =
results[i];
String sObject = result.getSubject();
System.out.println("sObject: " + sObject);
System.out.println("Group name: " + result.getName());
System.out.println("Group label: " + result.getLabel());
System.out.println("Group description: " +
result.getDescription());

// Get the top-level categories
DataCategory[] topCategories = result.getTopCategories();

// Iterate through the top level categories and retrieve
// some information
for (int j = 0; j < topCategories.length; j++) {
DataCategory topCategory = topCategories[j];
System.out.println("Category name: " +
topCategory.getName());
System.out.println("Category label: " +
topCategory.getLabel());
DataCategory [] childCategories =
topCategory.getChildCategories();
System.out.println("Child categories: ");
for (int k = 0; k < childCategories.length; k++) {
System.out.println("\t" + k + ". Category name: " +
childCategories[k].getName());
System.out.println("\t" + k + ". Category label: " +
childCategories[k].getLabel());
}
}
} catch (ConnectionException ce) {
ce.printStackTrace();
}
}
```

Sample Code—C#

This sample shows how to use `sObject` and data category group pairs to retrieve data categories for each pair. It calls `describeDataCategoryGroupStructures()` with two pairs, `KnowledgeArticleVersion/Regions` and `Question/Regions`, and iterates through the results of this call. It gets the top categories for each result, which is "All", and then gets the first-level child categories. The sample requires that you set up a data category group called *Regions* with some child categories and associate it with a knowledge article and questions. Alternatively, you can replace the data category group name in the sample if you want to use an existing data category group in your org that has a different name.

```
public void describeDataCategoryGroupStructuresSample() {
    try {
        // Create the data category pairs
        DataCategoryGroupObjectTypePair pair1 =
            new DataCategoryGroupObjectTypePair();
        DataCategoryGroupObjectTypePair pair2 =
            new DataCategoryGroupObjectTypePair();
        pair1.subject = "KnowledgeArticleVersion";
        //pair1.setDataCategoryGroupName("Regions");
        pair1.dataCategoryGroupName = "KBArticleCategories";
        pair2.subject = "Question";
        //pair2.setDataCategoryGroupName("Regions");
        pair2.dataCategoryGroupName = "KBArticleCategories";

        DataCategoryGroupObjectTypePair[] pairs =
            new DataCategoryGroupObjectTypePair[] {
                pair1,
                pair2
            };

        // Get the list of top level categories using the describe call
        DescribeDataCategoryGroupStructureResult[] results =
            binding.describeDataCategoryGroupStructures(
                pairs,
                false
            );

        // Iterate through each result and get some properties
        // including top categories and child categories
        for (int i = 0; i < results.Length; i++) {
            DescribeDataCategoryGroupStructureResult result =
                results[i];
            String sObject = result.subject;
            Console.WriteLine("sObject: " + sObject);
            Console.WriteLine("Group name: " + result.name);
            Console.WriteLine("Group label: " + result.label);
            Console.WriteLine("Group description: " +
                result.description);

            // Get the top-level categories
            DataCategory[] topCategories = result.topCategories;

            // Iterate through the top level categories and retrieve
            // some information
            for (int j = 0; j < topCategories.Length; j++) {
```

```

DataCategory topCategory = topCategories[j];
Console.WriteLine("Category name: " +
topCategory.name);
Console.WriteLine("Category label: " +
topCategory.label);
DataCategory [] childCategories =
topCategory.childCategories;
Console.WriteLine("Child categories: ");
for (int k = 0; k < childCategories.Length; k++) {
Console.WriteLine("\t" + k + ". Category name: " +
childCategories[k].name);
Console.WriteLine("\t" + k + ". Category label: " +
childCategories[k].label);
}
}
}
}
}
catch (SoapException e)
{
Console.WriteLine("An unexpected error has occurred: " +
e.Message + "\n" + e.StackTrace);
}
}
}

```

Arguments

Name	Type	Description
pairs	DataCategoryGroupObjectTypePair []	Specifies a category group and an object to query. Visible data categories are retrieved for that object.
topCategoriesOnly	boolean	Indicates whether the call returns only the top (<code>true</code>) or all the categories (<code>false</code>) visible depending on the user's data category group visibility settings. For more information on data category group visibility, see Data Category Visibility in the Salesforce online help.

[DataCategoryGroupObjectTypePair](#) contains the following fields:

Name	Type	Description
dataCategoryGroupName	string	The unique name used for API access to the data category group.
subject	string	The object associated with the data category group

Response

[describeDataCategoryGroupStructures\(\)](#) []

Faults

[InvalidObjectFault](#)

[UnexpectedErrorFault](#)

describeDataCategoryGroupStructures ()

The describeDataCategoryGroupStructures() call returns an array of DescribeDataCategoryGroupStructureResult objects containing the category groups and categories associated with the specified objects.

Name	Type	Description
description	string	The description of the data category group.
label	string	The label for the data category group in the Salesforce user interface.
name	string	The unique name used for API access to the data category group.
subject	string	The object associated with the data category group.
topCategories	DataCategory[]	A list of top level categories visible depending on the user's data category group visibility settings. For more information on data category group visibility, see "Data Category Visibility" in the Salesforce online help.

DataCategory

Name	Type	Description
childDataCategories	DataCategory[]	A recursive list of visible sub categories in the data category.
label	string	The label for the data category in the Salesforce user interface.
name	string	The unique name used for API access to the data category.

describeGlobal ()

Retrieves a list of available objects for your organization's data.

Syntax

```
DescribeGlobalResult = connection.describeGlobal();
```

Usage

Use [describeGlobal\(\)](#) to obtain a list of available objects for your organization. You can then iterate through this list and use [describeSObjects\(\)](#) to obtain metadata about individual objects.

Your client application must be logged in with sufficient access rights to retrieve metadata about your organization's data. For more information, see [Factors that Affect Data Access](#).

Sample Code—Java

This sample shows how to perform a global describe. It then retrieves the sObjects from the global describe result and writes their names to the console.

```
public void describeGlobalSample() {
    try {
        // Make the describeGlobal() call
        DescribeGlobalResult describeGlobalResult =
            connection.describeGlobal();

        // Get the sObjects from the describe global result
        DescribeGlobalSObjectResult[] subjectResults =
            describeGlobalResult.getSObjects();

        // Write the name of each sObject to the console
        for (int i = 0; i < subjectResults.length; i++) {
            System.out.println(subjectResults[i].getName());
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample shows how to perform a global describe. It then retrieves the sObjects from the global describe result and writes their names to the console.

```
public void describeGlobalSample()
{
    try
    {
        // Make the describeGlobal() call
        DescribeGlobalResult dgr = binding.describeGlobal();

        // Get the sObjects from the describe global result
        DescribeGlobalSObjectResult[] sObjResults = dgr.subjects;

        // Write the name of each sObject to the console
        for (int i = 0; i < sObjResults.Length; i++)
        {
            Console.WriteLine(sObjResults[i].name);
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

Arguments

None.

Response

[DescribeGlobalResult](#)

Fault

[UnexpectedErrorFault](#)

SEE ALSO:

[describeSObjects\(\)](#)

[API Call Basics](#)

[Using the Partner WSDL](#)

https://developer.salesforce.com/page/Sample_SOAP_Messages

DescribeGlobalResult

The `describeGlobal()` call returns a `DescribeGlobalResult` object, which has the following properties.

Name	Type	Description
<code>encoding</code>	<code>string</code>	Specifies how an org's data is encoded, such as UTF-8 or ISO-8859-1.
<code>maxBatchSize</code>	<code>int</code>	Maximum number of records allowed in a <code>create()</code> , <code>update()</code> , or <code>delete()</code> call.
<code>subjects</code>	DescribeGlobalSObjectResult[]	List of result objects that returns information about the available objects for your org. Available in API version 17.0 and later. This property enhances the information that was previously available in the <code>types</code> property.
<code>types</code>	<code>string[]</code>	List of available objects for your org. You iterate through this list to retrieve the object string that you pass to <code>describeSObjects()</code> . Beginning with API version 17.0, this property is no longer supported. Use the <code>name</code> property in <code>DescribeGlobalSObjectResult</code> instead.

DescribeGlobalSObjectResult

Represents the properties for one of the objects available for your org. Each object has the following properties:

Name	Type	Description
<code>activateable</code>	<code>boolean</code>	Reserved for future use.

Name	Type	Description
<code>createable</code>	boolean	Indicates whether the object can be created via the <code>create()</code> call (<code>true</code>) or not (<code>false</code>).
<code>custom</code>	boolean	Indicates whether the object is a custom object (<code>true</code>) or not (<code>false</code>).
<code>customSetting</code>	boolean	Indicates whether the object is a custom setting object (<code>true</code>) or not (<code>false</code>).
<code>dataTranslationEnabled</code>	boolean	Indicates whether data translation is enabled for the object (<code>true</code>) or not (<code>false</code>). Available in API version 49.0 and later.
<code>deepCloneable</code>	boolean	Reserved for future use.
<code>deletable</code>	boolean	Indicates whether the object can be deleted via the <code>delete()</code> call (<code>true</code>) or not (<code>false</code>).
<code>deprecatedAndHidden</code>	boolean	Reserved for future use.
<code>feedEnabled</code>	boolean	Indicates whether Chatter feeds are enabled for the object (<code>true</code>) or not (<code>false</code>). This property is available in API version 19.0 and later.
<code>isInterface</code>	boolean	Reserved for future use.
<code>keyPrefix</code>	string	<p>Three-character prefix code in the object ID. Object IDs are prefixed with three-character codes that specify the type of the object. For example, Account objects have a prefix of 001 and Opportunity objects have a prefix of 006. Note that a key prefix can sometimes be shared by multiple objects so it does not always uniquely identify an object.</p> <p>Use the value of this field to determine the object type of a parent in those cases where the child may have more than one object type as parent (polymorphic). For example, you may need to obtain the <code>keyPrefix</code> value for the parent of a Task or Event.</p>
<code>label</code>	string	Label text for a tab or field renamed in the user interface, if applicable, or the object name, if not. For example, an organization representing a medical vertical might rename <code>Account</code> to <code>Patient</code> . Tabs and fields can be renamed in the Salesforce user interface. See the Salesforce online help for more information.
<code>labelPlural</code>	string	Label text for an object that represents the plural version of an object name, for example, "Accounts."
<code>layoutable</code>	boolean	Indicates whether the object supports the <code>describeLayout()</code> call (<code>true</code>) or not (<code>false</code>).
<code>mergeable</code>	boolean	Indicates whether the object can be merged with other objects of its type (<code>true</code>) or not (<code>false</code>). <code>true</code> for leads, contacts, and accounts.
<code>mruEnabled</code>	boolean	Indicates whether Most Recently Used (MRU) list functionality is enabled for the object (<code>true</code>) or not (<code>false</code>).
<code>name</code>	string	Name of the object. This name is equivalent to an entry in the <code>types</code> list that is no longer supported, beginning with API version 17.0.

Name	Type	Description
queryable	boolean	Indicates whether the object can be queried via the <code>query()</code> call (<code>true</code>) or not (<code>false</code>).
replicateable	boolean	Indicates whether the object can be replicated via the <code>getUpdated()</code> and <code>getDeleted()</code> calls (<code>true</code>) or not (<code>false</code>).
retrieveable	boolean	Indicates whether the object can be retrieved via the <code>retrieve()</code> call (<code>true</code>) or not (<code>false</code>).
searchable	boolean	Indicates whether the object can be searched via the <code>search()</code> call (<code>true</code>) or not (<code>false</code>).
triggerable	boolean	Indicates whether the object supports Apex triggers.
undeletable	boolean	Indicates whether an object can be undeleted using the <code>undelete()</code> call (<code>true</code>) or not (<code>false</code>).
updateable	boolean	Indicates whether the object can be updated via the <code>update()</code> call (<code>true</code>) or not (<code>false</code>).

describeGlobalTheme ()

Returns information about both objects and themes available to the current logged-in user.

Syntax

```
DescribeGlobalTheme = connection.describeGlobalTheme ();
```

Usage

Use `describeGlobalTheme()` to get both a list of available objects and theme information about those objects for your organization. `describeGlobalTheme()` is a combination of `describeGlobal()` and `describeTheme()` combined into a single call.

Your client application must be logged in with sufficient access rights to retrieve theme and object information about your organization's data. For more information, see [Factors that Affect Data Access](#).

`describeGlobalTheme()` is available in API version 29.0 and later.

Sample

This Java sample calls `describeGlobalTheme()` and then iterates over the retrieved object and theme information.

```
public static void describeGlobalThemeExample () {
    try {
        // Get current theme and object information
        DescribeGlobalTheme globalThemeResult = connection.describeGlobalTheme ();
        DescribeGlobalResult globalResult = globalThemeResult.getGlobal ();
        DescribeThemeResult globalTheme = globalThemeResult.getTheme ();
```

```

// For the themes, get the array of theme items, one per object
DescribeThemeItem[] themeItems = globalTheme.getThemeItems();
for (int i = 0; i < themeItems.length; i++) {
    DescribeThemeItem themeItem = themeItems[i];
    System.out.println("Theme information for object " + themeItem.getName());
    // Get color and icon info for each themeItem
    DescribeColor colors[] = themeItem.getColors();
    System.out.println("    Number of colors: " + colors.length);
    int k;
    for (k = 0; k < colors.length; k++) {
        DescribeColor color = colors[k];
        System.out.println("        For Color #" + k + ":");
        System.out.println("            Web RGB Color: " + color.getColor());
        System.out.println("            Context: " + color.getContext());
        System.out.println("            Theme: " + color.getTheme());
    }
    DescribeIcon icons[] = themeItem.getIcons();
    System.out.println("    Number of icons: " + icons.length);
    for (k = 0; k < icons.length; k++) {
        DescribeIcon icon = icons[k];
        System.out.println("        For Icon #" + k + ":");
        System.out.println("            ContentType: " + icon.getContentType());
        System.out.println("            Height: " + icon.getHeight());
        System.out.println("            Theme: " + icon.getTheme());
        System.out.println("            URL: " + icon.getUrl());
        System.out.println("            Width: " + icon.getWidth());
    }
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

Response

[DescribeGlobalTheme](#)

Faults

[UnexpectedErrorFault](#)

SEE ALSO:

[DescribeGlobalTheme](#)

[DescribeThemeResult](#)

[DescribeThemeItem](#)

[DescribeColor](#)

[DescribeIcon](#)

DescribeGlobalTheme

The `describeGlobalTheme()` call returns `DescribeGlobalTheme`, which contains a `DescribeThemeResult` and a `DescribeGlobalResult`.

Name	Type	Description
global	DescribeGlobalResult	Object information.
theme	DescribeThemeResult	Theme information.

describeKnowledge()

Retrieves the Knowledge language settings in the organization.

Syntax

```
KnowledgeSettings result = _connection.describeKnowledgeSettings();
```

Usage

Use this call to describe the existing Knowledge language settings, including the default Knowledge language, supported languages, and a list of Knowledge language information. You can also use `KnowledgeSettings` in the Metadata API to obtain similar information.

Sample Code—Java

This sample shows how to retrieve the Knowledge language settings. It returns the default Knowledge language, a list of Knowledge supported language, including the language code and whether it's an active Knowledge language.

```
public void describeKnowledgeSettingsSample() {
    try {

        // Make the describe call for KnowledgeSettings
        KnowledgeSettings result = connection.describeKnowledgeSettings();

        // Get the properties of KnowledgeSettings
        System.out.println("Knowledge default language: " + result.getDefaultLanguage());
        for (KnowledgeLanguageItem lang : result.getLanguages()) {
            System.out.println("Language: " + lang.getName());
            System.out.println("Active: " + lang.isActive());
        }
    } catch (ConnectionException ex) {
        ex.printStackTrace();
    }
}
```

Sample Code—C#

This sample shows how to retrieve the Knowledge language settings. It returns the default Knowledge language, a list of Knowledge supported language, including the language code and whether it's an active Knowledge language.

```
public void describeKnowledgeSettingsSample() {
    try {

        // Make the describe call for KnowledgeSettings
        KnowledgeSettings result = connection.describeKnowledgeSettings();

        // Get the properties of KnowledgeSettings
        Console.WriteLine("Knowledge default language: " + result.getDefaultLanguage());
        for (KnowledgeLanguageItem lang : result.getLanguages()) {
            Console.WriteLine("Language: " + lang.getName());
            Console.WriteLine("Active: " + lang.isActive());
        }
    } catch (SoapException ex) {
        ex.printStackTrace();
    }
}
```

Response

KnowledgeSettings

describeLayout()

Retrieves metadata about page layouts for the specified object type.

Syntax

```
DescribeLayoutResult = connection.describeLayout(string sObjectType, string layoutName,
ID recordTypeID[]);
```


Usage

Use this call to retrieve information about the layout (presentation of data to users) for a given object type. This call returns metadata about a given page layout, such as the detail page layout, the edit page layout, and the record type mappings. For additional information, see "Page Layouts" in Salesforce Help .

Generally, user profiles have one layout associated with each object. In Enterprise, Unlimited, and Performance Editions, user profiles can have multiple layouts per object, where each layout is specific to a given record type. This call returns metadata for multiple layouts, if applicable.

Layouts can be further customized in standard objects that have defined named layouts, which are separate from the primary layout for both the profile and the record type. One example of named layouts is the UserAlt layout defined on the User object, which is consumed in the Salesforce mobile app instead of the primary User layout. New layout names can only be defined by Salesforce, but customization of named layouts is controlled by administrators in the same way as primary layouts.

If you supply a null value for `recordTypeIds`, all the layouts for that user are returned, instead of just the layouts for each specified record type. The same layout can be associated with multiple record types for the user's profile, in which case there would only be one layout returned.

 **Note:** This call is an advanced API call that is typically used only by partners who have written custom page rendering code for generating output on a specialized device (for example, on PDAs) and need to examine the layout details of an object before rendering the page output.

Use the following procedure to describe layouts:

1. To display a detail page or edit page for a record that exists, a client application first gets the `recordTypeIds` from the record, then it finds the `layoutId` associated with that `recordTypeIds` (through `recordTypeMapping`), and finally it uses that layout information to render the page.
2. To display the create version of an edit page, a client application first determines whether more than one record type is available and, if so, presents the user with a choice. Once a record type has been chosen, then the client application uses the layout information to render the page. It uses the picklist values from the `RecordTypeMapping` to display valid picklist values for picklist fields.
3. A client application can access the labels for the layout, using the `DescribeLayoutResult`.

The following restrictions apply to person account record types:

- `describeLayout()` for version 7.0 and below returns the default business account record type as the default record type even if the tab default is a person account record type. In version 8.0 and after, it will always be the tab default.
- `describeLayout()` for version 7.0 and below doesn't return any person account record types.

For more information about person account record types, see [Person Account Record Types](#).

Sample Code—Java

This sample shows how to get the layouts of an Account sObject. It calls `describeLayout()` with the name of the sObject type to describe. It doesn't specify record type IDs as a third argument, which means that layouts for all record types will be returned if record types are defined in your org for the specified sObject. After getting the layout, the sample writes the number of detail and edit sections found and their headings. Next, it iterates through each edit layout section and retrieves its components.

```
public void describeLayoutSample() {
    try {
        String objectToDescribe = "Account";
        DescribeLayoutResult dlr =
            connection.describeLayout(objectToDescribe, null, null);
        System.out.println("There are " + dlr.getLayouts().length +
            " layouts for the " + objectToDescribe + " object."
        );

        // Get all the layouts for the sObject
        for(int i = 0; i < dlr.getLayouts().length; i++) {
            DescribeLayout layout = dlr.getLayouts()[i];
            DescribeLayoutSection[] detailLayoutSectionList =
                layout.getDetailLayoutSections();
            System.out.println(" There are " +
                detailLayoutSectionList.length +
                " detail layout sections");
            DescribeLayoutSection[] editLayoutSectionList =
                layout.getEditLayoutSections();
            System.out.println(" There are " +
                editLayoutSectionList.length +
```

```

        " edit layout sections");

// Write the headings of the detail layout sections
for(int j = 0; j < detailLayoutSectionList.length; j++) {
    System.out.println(j +
        " This detail layout section has a heading of " +
        detailLayoutSectionList[j].getHeading());
}

// Write the headings of the edit layout sections
for(int x = 0; x < editLayoutSectionList.length; x++) {
    System.out.println(x +
        " This edit layout section has a heading of " +
        editLayoutSectionList[x].getHeading());
}

// For each edit layout section, get its details.
for(int k = 0; k < editLayoutSectionList.length; k++) {
    DescribeLayoutSection els =
        editLayoutSectionList[k];
    System.out.println("Edit layout section heading: " +
        els.getHeading());
    DescribeLayoutRow[] dlrList = els.getLayoutRows();
    System.out.println("This edit layout section has " +
        dlrList.length + " layout rows.");
    for(int m = 0; m < dlrList.length; m++) {
        DescribeLayoutRow lr = dlrList[m];
        System.out.println(" This row has " +
            lr.getNumItems() + " layout items.");
        DescribeLayoutItem[] dliList = lr.getLayoutItems();
        for(int n = 0; n < dliList.length; n++) {
            DescribeLayoutItem li = dliList[n];
            if ((li.getLayoutComponents() != null) &&
                (li.getLayoutComponents().length > 0)) {
                System.out.println("\tLayout item " + n +
                    ", layout component: " +
                    li.getLayoutComponents()[0].getValue());
            }
            else {
                System.out.println("\tLayout item " + n +
                    ", no layout component");
            }
        }
    }
}

// Get record type mappings
if (dlr.getRecordTypeMappings() != null) {
    System.out.println("There are " +
        dlr.getRecordTypeMappings().length +
        " record type mappings for the " +
        objectToDescribe + " object"
    );
}

```

```

    } else {
        System.out.println(
            "There are no record type mappings for the " +
            objectToDescribe + " object."
        );
    }
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

Sample Code—C#

This sample shows how to get the layouts of an Account sObject. It calls `describeLayout()` with the name of the sObject type to describe. It doesn't specify record type IDs as a third argument, which means that layouts for all record types will be returned if record types are defined in your org for the specified sObject. After getting the layout, the sample writes the number of detail and edit sections found and their headings. Next, it iterates through each edit layout section and retrieves its components.

```

public void describeLayoutSample()
{
    try
    {
        String objectToDescribe = "Account";
        DescribeLayoutResult dlr =
            binding.describeLayout(objectToDescribe, null, null);
        Console.WriteLine("There are " + dlr.layouts.Length +
            " layouts for the " + objectToDescribe + " object."
        );

        // Get all the layouts for the sObject
        for (int i = 0; i < dlr.layouts.Length; i++)
        {
            DescribeLayout layout = dlr.layouts[i];
            DescribeLayoutSection[] detailLayoutSectionList =
                layout.detailLayoutSections;
            Console.WriteLine(" There are " +
                detailLayoutSectionList.Length +
                " detail layout sections");
            DescribeLayoutSection[] editLayoutSectionList =
                layout.editLayoutSections;
            Console.WriteLine(" There are " +
                editLayoutSectionList.Length +
                " edit layout sections");

            // Write the headings of the detail layout sections
            for (int j = 0; j < detailLayoutSectionList.Length; j++)
            {
                Console.WriteLine(j +
                    " This detail layout section has a heading of " +
                    detailLayoutSectionList[j].heading);
            }

            // Write the headings of the edit layout sections

```

```

for (int x = 0; x < editLayoutSectionList.Length; x++)
{
    Console.WriteLine(x +
        " This edit layout section has a heading of " +
        editLayoutSectionList[x].heading);
}

// For each edit layout, get its details.
for (int k = 0; k < editLayoutSectionList.Length; k++)
{
    DescribeLayoutSection els =
        editLayoutSectionList[k];
    Console.WriteLine("Edit layout section heading: " +
        els.heading);
    DescribeLayoutRow[] dlrList = els.layoutRows;
    Console.WriteLine("This edit layout section has " +
        dlrList.Length + " layout rows.");
    for (int m = 0; m < dlrList.Length; m++)
    {
        DescribeLayoutRow lr = dlrList[m];
        Console.WriteLine(" This row has " +
            lr.numItems + " layout items.");
        DescribeLayoutItem[] dliList = lr.layoutItems;
        for (int n = 0; n < dliList.Length; n++)
        {
            DescribeLayoutItem li = dliList[n];
            if ((li.layoutComponents != null) &&
                (li.layoutComponents.Length > 0))
            {
                Console.WriteLine("\tLayout item " + n +
                    ", layout component: " +
                    li.layoutComponents[0].value);
            }
            else
            {
                Console.WriteLine("\tLayout item " + n +
                    ", no layout component");
            }
        }
    }
}

// Get record type mappings
if (dlr.recordTypeMappings != null)
{
    Console.WriteLine("There are " +
        dlr.recordTypeMappings.Length +
        " record type mappings for the " +
        objectToDescribe + " object");
}
else
{
    Console.WriteLine(
        "There are no record type mappings for the " +

```

```

        objectToDescribe + " object.");
    }
}
}
catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}
}

```

Arguments

Name	Type	Description
sObjectType	string	The specified value must be a valid object for your organization. If the object is a person account, specify Account, or if it is a person contact, specify Contact.
layoutName	string	The specified value must be a valid named layout for this object. Layout names are obtained from <code>namedLayoutInfos</code> in <code>DescribeSObjectResult</code> . The entity name is not valid because the primary layout is not considered "named."
recordTypeIds	ID[]	Optional parameter restricts the layout data returned to the specified record types. To retrieve the layout for the primary record type, specify the value <code>012000000000000AAA</code> for the <code>recordTypeIds</code> regardless of the object. This value is returned in the <code>recordTypeInfo</code> s for the primary record type in the <code>DescribeSObjectResult</code> . A SOQL query returns a null value, not <code>012000000000000AAA</code> . For information on IDs, see ID Field Type .

Response

[DescribeLayoutResult](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[API Call Basics](#)

https://developer.salesforce.com/page/Sample_SOAP_Messages

DescribeLayoutResult

The `describeLayout()` call returns a `DescribeLayoutResult` object containing top-level record type information about the passed-in `sObjectType`, as well as a mapping of record types to layouts. Your client application can traverse this object to retrieve detailed metadata about the layout.



Tip: If you have actions in the publisher enabled in your organization, you can retrieve the layout definition for a global publisher layout by using `Global` as the `sObjectType` and `null` as the `recordTypeId`.

Name	Type	Description
<code>feedView</code>	DescribeLayoutFeedView[]	Feed view related layout data for a feed-based layout. This field is null for page layouts that are not feed-based.
<code>layouts</code>	DescribeLayout[]	Layout(s) associated with the specified <code>sObjectType</code> . In general, there is a one-to-one correspondence between layouts and objects. However, in some cases, an object will have multiple layouts in the context of a given user profile.
<code>recordTypeMappings</code>	RecordTypeMapping[]	Record type mapping(s) available for the user. The objects on a user profile can have multiple record types. All record types are returned, not just those available to the calling user. This allows the client application to display a layout appropriate for a given user profile. For example, suppose User A owns a record, and this record has record type X set. If User B tries to view this record, then the client application can display the record using the layout associated with this record type for User B's profile (even if the record type is not available for the user).
<code>recordTypeSelectorRequired</code>	boolean	If <code>true</code> , a record type selector page is required; if <code>false</code> , use the default record type.

DescribeLayout

Represents a specific layout for the specified `sObjectType`. Each `DescribeLayout` is referenced by its unique layout ID and consists of two types of views (represented in this object as arrays of [DescribeLayoutSection](#)):

- **Detail view**—Read-only display of the object. In a detail layout, certain pieces of information (such as address details) might be aggregated into a single [DescribeLayoutItem](#).
- **Edit view**—Editable display of the object. In an edit layout, individual pieces of information (such as an address) will be broken up into separate fields.

An individual `DescribeLayout` consists of these fields:

Name	Type	Description
<code>buttonLayoutSection</code>	DescribeLayoutButtonSection	Standard and custom button sections associated with the specified layout.
<code>detailLayoutSections</code>	DescribeLayoutSection[]	Layout section(s) for the detail view.
<code>editLayoutSections</code>	DescribeLayoutSection[]	Layout section(s) for the edit view.

Name	Type	Description
highlightsPanelLayoutSection	DescribeLayoutSection []	Layout section(s) for the highlights panel view.
multirowEditLayoutSections	DescribeLayoutSection []	Layout section(s) for the multiline layout view. This field is available in API version 35.0 and later.
id	ID	Unique ID of this layout. For information on IDs, see ID Field Type .
quickActionList	DescribeQuickActionListResult	List of actions associated with the specified layout. This field is available in API version 28.0 and later.
relatedContent	RelatedContent	Mobile Cards section associated with the specified layout. This field is available in API version 29.0 and later.
relatedLists	RelatedList []	Related list(s) associated with the specified layout.
saveOptions	DescribeLayoutSaveOption []	List of save options for the layout.

DescribeLayoutButtonSection

Represents one of two sections of the layout containing either standard or custom buttons.

Name	Type	Description
detailButtons	DescribeLayoutButton []	Standard or custom button(s) associated with the specified button section.

DescribeLayoutButton

Represents a single standard button, custom button, or custom link in a [DescribeLayout](#).

Name	Type	Description
behavior	WebLinkWindowType	What the button or link does when clicked, such as execute JavaScript or open its content source in a new window, for example. This field is available in API version 31.0 and later.
colors	DescribeColor []	Array of color information for icons associated with this button or link. Each color is associated with a theme. This field is available in API version 32.0 and later.
content	string	The API name of the Visualforce page or s-control being delivered. This field is available in API version 31.0 and later.
contentSource	WebLinkType	The content source of the custom button or link. The <code>contentSource</code> for a standard button which hasn't been overridden is <code>null</code> . This field is available in API version 31.0 and later.

Name	Type	Description
<code>custom</code>	boolean	Required. Indicates whether this is a custom button or link (<code>true</code>) or not (<code>false</code>).
<code>encoding</code>	string	Type of encoding assigned to the URL called by the button or link. Valid values are: <ul style="list-style-type: none"> • <code>UTF-8</code>—Unicode (UTF-8) • <code>ISO-8859-1</code>—General US & Western Europe (ISO-8859-1, ISO-LATIN-1) • <code>Shift_JIS</code>—Japanese (Shift-JIS) • <code>ISO-2022-JP</code>—Japanese (JIS) • <code>EUC-JP</code>—Japanese (EUC-JP) • <code>x-SJIS_0213</code>—Japanese (Shift-JIS_2004) • <code>ks_c_5601-1987</code>—Korean (ks_c_5601-1987) • <code>Big5</code>—Traditional Chinese (Big5) • <code>GB2312</code>—Simplified Chinese (GB2312) • <code>Big5-HKSCS</code>—Traditional Chinese Hong Kong (Big5-HKSCS) This field is available in API version 31.0 and later.
<code>height</code>	int	The height (in pixels) when a button or link's <code>behavior</code> field value is set to <code>newWindow</code> , <code>sidebar</code> , or <code>noSidebar</code> . This field is available in API version 31.0 and later.
<code>icons</code>	DescribeIcon []	Array of icons for this button or link. Each icon is associated with a theme. This field is available in API version 29.0 and later.
<code>label</code>	string	Label for the button or link displayed in the Salesforce user interface.
<code>menubar</code>	boolean	Indicates whether the menu bar displays (<code>true</code>) or not (<code>false</code>) when a button or link's <code>behavior</code> field value is set to <code>newWindow</code> . This field is available in API version 31.0 and later.
<code>name</code>	string	API name of the button or link.
<code>overridden</code>	boolean	Required. Indicates whether a standard button has been overridden (<code>true</code>) or not (<code>false</code>). This field is available in API version 31.0 and later.
<code>resizeable</code>	boolean	Indicates whether the new window is resizeable (<code>true</code>) or not (<code>false</code>) when a button or link's <code>behavior</code> field value is set to <code>newWindow</code> . This field is available in API version 31.0 and later.

Name	Type	Description
<code>scrollbars</code>	boolean	Indicates whether scrollbars display (<code>true</code>) or not (<code>false</code>) when a button or link's <code>behavior</code> field value is set to <code>newWindow</code> . This field is available in API version 31.0 and later.
<code>showsLocation</code>	boolean	Indicates whether the address bar displays (<code>true</code>) or not (<code>false</code>) when a button or link's <code>behavior</code> field value is set to <code>newWindow</code> . This field is available in API version 31.0 and later.
<code>showsStatus</code>	boolean	Indicates whether the status bar displays (<code>true</code>) or not (<code>false</code>) when a button or link's <code>behavior</code> field value is set to <code>newWindow</code> . This field is available in API version 31.0 and later.
<code>toolbar</code>	boolean	Indicates whether the toolbars display (<code>true</code>) or not (<code>false</code>) when a button or link's <code>behavior</code> field value is set to <code>newWindow</code> . This field is available in API version 31.0 and later.
<code>url</code>	string	The URL called by the button or link. This field is <code>null</code> for standard buttons in a related list. This field is available in API version 31.0 and later.
<code>width</code>	int	The width (in pixels) when a button or link's <code>behavior</code> field value is set to <code>newWindow</code> . This field is available in API version 31.0 and later.
<code>windowPosition</code>	WebLinkPosition	Indicates the window position when a button or link's <code>behavior</code> field value is set to <code>newWindow</code> . This field is available in API version 31.0 and later.

DescribeLayoutComponent

Represents the smallest unit in a layout—a field or a separator. To reference a field for display, a client application uses the following notation to reference a field in the [describeSObjects\(\)](#) call: `LayoutComponent.fieldName`.

In API version 31.0 and later, `DescribeLayoutComponent` is extended with [FieldLayoutComponent](#) if both the [LayoutComponentType](#) value is `Field`, and the field being described is either the compound field `Address` or the compound field `Person Name`.

Name	Type	Description
<code>displayLines</code>	int	The number of vertical lines displayed for a field in the edit view. Applies to <code>textarea</code> and multi-select picklist fields.
<code>tabOrder</code>	int	Indicates the tab order for the item in the row.

Name	Type	Description
<code>type</code>	<code>LayoutComponentType</code>	The <code>LayoutComponentType</code> for this <code>LayoutComponent</code> .
<code>value</code>	<code>string</code>	Value of this <code>LayoutComponent</code> . The name of the field if the <code>LayoutComponentType</code> value is <code>Field</code> . The API name of the canvas app if the <code>LayoutComponentType</code> value is <code>Canvas</code> .

DescribeLayoutFeedFilter

Represents an individual feed filter option that you can use to filter the feed.

Name	Type	Description
<code>label</code>	<code>string</code>	The label of the filter.
<code>name</code>	<code>string</code>	The API name of the filter.
<code>type</code>	<code>FeedLayoutFilterType</code> enum	Standard feed filter types: <ul style="list-style-type: none"> • <code>AllUpdates</code> • <code>FeedItemType</code>

DescribeLayoutFeedView

Represents the layout of the feed view for a feed-based page layout.

Name	Type	Description
<code>feedFilters</code>	<code>DescribeLayoutFeedFilter[]</code>	Lists the feed filter options that are displayed with the feed.

DescribeLayoutItem

Represents an individual item in a `DescribeLayoutRow`. A `DescribeLayoutItem` consists of a set of components (`DescribeLayoutComponent`), each of which is either a field or a separator. For most fields on a layout, there is only one component per layout item. However, in a display-only view, the `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 `DescribeLayoutItems`.

Name	Type	Description
<code>editable</code>	<code>boolean</code>	Indicates whether this <code>DescribeLayoutItem</code> can be edited (<code>true</code>) or not (<code>false</code>). This field is available in API version 30.0 and below. It was replaced by the <code>editableForNew</code> and <code>editableForUpdate</code> fields in API version 31.0.
<code>editableForNew</code>	<code>boolean</code>	Indicates whether a new <code>DescribeLayoutItem</code> can be edited when creating a new record (<code>true</code>) or not (<code>false</code>). This field is available in API version 31.0 and later.

Name	Type	Description
<code>editableForUpdate</code>	boolean	Indicates whether an existing <code>DescribeLayoutItem</code> can be edited when editing a record (<code>true</code>) or not (<code>false</code>). This field is available in API version 31.0 and later.
<code>label</code>	string	Label text for this <code>DescribeLayoutItem</code> .
<code>layoutComponents</code>	<code>DescribeLayoutComponent[]</code>	DescribeLayoutComponent for this <code>DescribeLayoutItem</code> .
<code>placeholder</code>	boolean	Indicates whether this <code>DescribeLayoutItem</code> is a placeholder (<code>true</code>) or not (<code>false</code>). If <code>true</code> , then this <code>DescribeLayoutItem</code> is blank.
<code>required</code>	boolean	Indicates whether this <code>DescribeLayoutItem</code> is required (<code>true</code>) or not (<code>false</code>). This is useful to know if, for example, you wanted to render required fields in a contrasting color (such as red).

DescribeLayoutRow

Represents a row in a [DescribeLayoutSection](#). A `DescribeLayoutRow` consists of one or more [DescribeLayoutItem](#) objects. For each `DescribeLayoutRow`, a `DescribeLayoutItem` refers either to a specific field or to an “empty” `DescribeLayoutItem` (a `DescribeLayoutItem` that contains no [DescribeLayoutComponent](#) objects). An empty `DescribeLayoutItem` can be returned when a given `DescribeLayoutRow` is sparse (for example, containing more fields on the right column than on the left column). Where there are gaps in the layout, an empty `DescribeLayoutItem` is returned as a placeholder.

Name	Type	Description
<code>layoutItems</code>	DescribeLayoutItem []	Refers to either a specific field or to an empty <code>LayoutItem</code> (a <code>LayoutItem</code> that contains no DescribeLayoutComponent objects).
<code>numItems</code>	int	Number of <code>layoutItems</code> . This information is redundant but, due to a bug in a popular SOAP toolkit, was required to avoid serialization problems.

DescribeLayoutSection

Represents a section of a [DescribeLayout](#) and consists of one or more columns and one or more rows (an array of [DescribeLayoutRow](#)).

Name	Type	Description
<code>columns</code>	int	Number of columns in this <code>DescribeLayoutSection</code> .
<code>heading</code>	string	Heading text (label) for this <code>DescribeLayoutSection</code> .
<code>layoutRows</code>	<code>DescribeLayoutRow</code> []	Array of one or more DescribeLayoutRow objects.
<code>parentLayoutId</code>	ID	The ID of the layout upon which this <code>DescribeLayoutSection</code> resides. This field is available in API version 35.0 and later.
<code>rows</code>	int	Number of rows in this <code>DescribeLayoutSection</code> .

Name	Type	Description
tabOrder	string	Indicates the tab order for the fields in the section in the edit view. Valid values are: <ul style="list-style-type: none"> LeftToRight TopToBottom This field is available in API version 31.0 and later.
useCollapsibleSection	boolean	Indicates whether this DescribeLayoutSection is a collapsible section, also known as a “twistie” (<code>true</code>), or not (<code>false</code>).
useHeading	boolean	Indicates whether to display the heading (<code>true</code>) or not (<code>false</code>).

DescribeQuickActionResult

Represents a list of actions assigned to the page layout. Available in API version 28.0 and later.

Name	Type	Description
quickActionListItems	DescribeQuickActionResult[]	Array of zero or more QuickActionListItemResult objects.

DescribeQuickActionListItemResult

Represents a QuickAction assigned to the actions list for a page layout. Available in API version 28.0 and later.

Name	Type	Description
colors	DescribeColor[]	Array of color information. Each color is associated with a theme. This field is available in API version 29.0 and later.
iconUrl	string	The URL of the icon associated with the action. This icon URL corresponds to the 32x32 icon used for the current Salesforce theme, introduced in Spring '10.
icons	DescribeIcon[]	Array of icons for this action. Each icon is associated with a theme. This field is available in API version 29.0 and later.
label	string	The label of the action.
miniIconUrl	string	The URL of the mini icon associated with the action. This icon URL corresponds to the 16x16 icon used for the current Salesforce theme, introduced in Spring '10.
quickActionName	string	The API name of the action.
targetObjectType	string	The API name of the action's target object.
type	string	The QuickActionType of the action. Valid values are: <ul style="list-style-type: none"> Create VisualforcePage

CustomLinkComponent

When the [LayoutComponentType](#) value is `CustomLink`, this type contains information about a single custom link on the page layout.

Name	Type	Description
<code>customLink</code>	DescribeLayoutButton	A single <code>LayoutComponent</code> object of type <code>CustomLink</code> .

FieldLayoutComponent

Extends the information returned by `describeLayoutComponent`. When the [LayoutComponentType](#) value is `Field`, and the field being described is an `Address` or `Person Name` field, `FieldLayoutComponent` includes information about the field's components. When the [LayoutComponentType](#) value is `Field`, and the field being described is a compound field, such as `Address` or `Person Name`, `FieldLayoutComponent` includes information about its components.

Available in API version 31.0 and later.

Name	Type	Description
<code>components</code>	<code>describeLayoutComponent[]</code>	Array of zero or more <code>LayoutComponent</code> objects of type <code>Field</code> .
<code>fieldType</code>	FieldType	The field type.

Sample Code for Usage of FieldLayoutComponent

```
DescribeLayoutComponent layoutComponent = layoutComponents[n];
// Look for a component representing the BillingAddress field
if (layoutComponent.getType() == LayoutComponentType.Field.toString() &&
    layoutComponent.getValue().equals("BillingAddress")) {
    // Cast this component as a FieldLayoutComponent
    DescribeLayoutComponent.FieldLayoutComponent addressFieldComponent =
(FieldLayoutComponent) layoutComponent;
    // At this point you can access addressFieldComponent
    FieldLayoutComponent-specific methods such as getComponents() or
    getFieldTypeInfo()
}
```

LayoutComponentType

Represents the type for a [DescribeLayoutComponent](#). Contains one of these values:

- `AnalyticsCloud`—A CRM Analytics dashboard on the page layout. Available in API version 34.0 and later.
- `Canvas`—A canvas component on the page layout. This layout component type is available in API version 31.0 and later.
- `CustomLink`—A custom link on the page layout.
- `EmptySpace`—A blank space on the page layout.
- `ExpandedLookup`—An Expanded Lookup component in the Mobile Cards section of the page layout.
- `Field`—Field name. A mapping to the [RecordTypeInfo](#) field on the [describeSObjectResult](#).
- `ReportChart`—A report chart on the page layout.

- `SControl`—Reserved for future use.
- `Separator`—Separator character, such as a semicolon (;) or slash (/).
- `VisualforcePage`—A Visualforce component on the page layout.

PicklistForRecordType


Represents a single record type picklist in a [RecordTypeMapping](#). The `picklistName` matches up with the `name` attribute of each field in the `fields` array in [describeSObjectResult](#). The `picklistValues` are the set of acceptable values for the `recordType`.

Name	Type	Description
<code>picklistName</code>	string	Name of the picklist.
<code>picklistValues</code>	PicklistEntry[]	Set of picklist values associated with the <code>recordTypeIds</code> in the RecordTypeMapping . Note: If you retrieve <code>picklistValues</code> , the PicklistEntry value is null. If you need the PicklistEntry value, get it from the PicklistEntry object obtained from the <code>Field</code> object associated with the DescribeSObjectResult .

RecordTypeMapping

Represents a single record type mapping in the `recordTypeMappings` field in a [DescribeLayoutResult](#) object. This object is a map of valid `recordTypeIds` to `layoutId`. For displaying a detail view, a client application uses this mapping to determine which layout is associated with the record type on the record. For displaying an edit view, a client application uses this mapping to determine which layout to use (and possibly to allow the user to choose between multiple record types); it will also determine the set of available picklist values.

Name	Type	Description
<code>available</code>	boolean	Indicates whether this record type is available (<code>true</code>) or not (<code>false</code>). Availability is used to display a list of available record types to the user when they are creating a new record.
<code>defaultRecordTypeMapping</code>	boolean	Indicates whether this is the default record type mapping (<code>true</code>) or not (<code>false</code>).
<code>layoutId</code>	ID	ID of the layout associated with this record type.
<code>name</code>	string	Name of this record type.
<code>picklistsForRecordType</code>	PicklistForRecordType []	Record type picklist(s) mapped to the <code>recordTypeIds</code> .
<code>recordTypeId</code>	ID	ID of this record type.

 **Note:** Some fields previously in this result have moved to [RecordTypeInfo](#) on page 287.

RelatedContent

Represents the Mobile Cards section in a [DescribeLayout](#). Available in API version 29.0 and later.

Name	Type	Description
<code>relatedContentItems</code>	DescribeRelatedContentItem []	An array of items in the Mobile Cards section of the page layout.

DescribeRelatedContentItem

Represents an individual item in the [DescribeRelatedContentItem](#) list. Available in API version 29.0 and later.

Name	Type	Description
<code>describeLayoutItem</code>	DescribeLayoutItem	An individual layout item in the Mobile Cards section. Must be wrapped in a DescribeRelatedContentItem to be added to the Mobile Cards section.

RelatedList

Represents a single related list in a [DescribeLayoutResult](#).

Name	Type	Description
<code>buttons</code>	DescribeLayoutButton []	Buttons associated with this related list. This field is available in API version 32.0 and later.
<code>columns</code>	RelatedListColumn []	Columns associated with this related list. You can pair this value with Field to achieve a number of useful tasks, including determining whether the field is: <ul style="list-style-type: none"> • A name field, in order to present a link to the detail • Sortable, (to allow the user to include it in an <code>ORDER BY</code> clause to sort the rows by the given column • A currency field, to include the currency symbol or code
<code>custom</code>	boolean	If <code>true</code> , this related list is custom.
<code>field</code>	string	Name of the field on the related (associated) object that establishes the relationship with the associating object. For example, for the Contact related list on Account , the value is <code>ACCOUNTID</code> .
<code>label</code>	string	Label for the related list, displayed in the Salesforce user interface.
<code>limitRows</code>	int	Number of rows to display.
<code>name</code>	string	Name of the ChildRelationship in the DescribeSObjectResult for the <code>sObjectType</code> which was provided as the argument to DescribeLayout .
<code>subject</code>	string	Name of the <code>sObjectType</code> that is the row type for rows within this related list.

Name	Type	Description
sort	RelatedListSort[]	If not null, the column(s) that should be used to order the related objects.

RelatedListColumn

Represents a single field in a related list returned by DescribeLayoutResult.

Name	Type	Description
field	string	API name of the field. This value is always of the form <i>object_type.field_name</i> . For example, if name is <code>Contact.Account.Owner.Alias</code> , then this value is <code>User.Alias</code> .
fieldApiName	string	SOQL field syntax for the field in relation to the main sObject for the related list. This value is always of the form <i>object_type.field_name</i> . Unlike <code>name</code> , it doesn't return a value in the Translate Returned SOQL Results format.
format	string	Display in date or dateTime format.
label	string	Label of the field.
lookupId	string	Optional SOQL field syntax to retrieve the lookup ID value for the main related list sObject . This value may be an expression that uses SOQL relationship query dot notation. For example, if the related list <code>sObjectType</code> is <code>Case</code> and the column display value is <code>Owner.Alias</code> , then the lookup ID value would be <code>Owner.Id</code> .
name	string	SOQL field syntax for the field in relation to the main sObject for the related list. This value may be an expression that uses SOQL relationship query dot notation, or it may use the Translate Returned SOQL Results or <code>convertCurrency()</code> format. For example, if the related list <code>sObjectType</code> is <code>Case</code> , then the value might be <code>Owner.Alias</code> or it might be <code>toLabel(Case.Status)</code> .

RelatedListSort

Represents the sorting preference for objects in the related list.

Name	Type	Description
column	string	Name of the field that is used to order the related objects.
ascending	boolean	If <code>true</code> , sort order is ascending. If <code>false</code> , descending.

Although in most cases there is only one `RelatedListSort` in the array, for some special standard related lists, there is more than one. If there is more than one, the `RelatedListSorts` are ordered according to how they should be included in a corresponding SOQL query, for example:

```
ORDER BY relatedListSort[0].getColumn() DIRECTION, relatedListSort[1].getColumn() DIRECTION
```


DescribeLayoutSaveOption

Represents the save options for the layout. Save options define behavior that occurs when objects are created or modified using the given layout. For example, for Cases and Leads, a “UseDefaultAssignmentRule” save option is exposed to control whether assignment rules are applied when Cases or Leads are created or edited.

Name	Type	Description
<code>defaultValue</code>	boolean	Default value for the save option. Controls whether the save option defaults to enabled or not in the Salesforce user interface. For example, for the “UseDefaultAssignmentRule” save option, if <code>defaultValue</code> is <code>true</code> , then by default the system triggers the default assignment rules when an Account, Case, or Lead is created or edited. If <code>false</code> , then the default assignment rules aren’t applied when an Account, Case, or Lead is created or edited, unless the user enables the save option in the Salesforce user interface.
<code>isDisplayed</code>	boolean	If <code>true</code> , then the save option is displayed in the layout. If <code>false</code> , then the save option isn’t displayed in the layout.
<code>label</code>	string	Label for the save option that is displayed in the Salesforce user interface.
<code>name</code>	string	API name for the save option.
<code>restHeaderName</code>	string	The corresponding REST API header for the save option.
<code>soapHeaderName</code>	string	The corresponding SOAP API header for the save option.

WebLinkPosition

Represents the window position for a new window opened upon clicking a [DescribeLayoutButton](#). Applies only to custom buttons. Available in API version 31.0 and later. Contains one of these values:

- `fullScreen`—The new window opens in a full screen. If this option is selected, any width or height parameters set for the new window are ignored.
- `none`—No window position preference is set.
- `topLeft`—The new window opens, positioned at the top left of the screen.

WebLinkType

Represents the content being delivered by the custom button. Contains one of these values:

- javascript
- page—Visualforce page
- sControl
- url

WebLinkWindowType

Represents the behavior for a [DescribeLayoutButton](#). Applies only to custom buttons. Available in API version 31.0 and later. Contains one of these values:

- newWindow—The custom button's content opens in a new browser window.
- noSidebar—The custom button's content displays in the existing browser window without a sidebar.
- onClickJavaScript—Valid only when the [DescribeLayoutButton](#)'s contentSource field value is javascript. Clicking the button or link executes JavaScript.
- replace—The custom button's content displays in the existing browser window without a sidebar or header.
- sidebar—The custom button's content displays in the existing browser window with a sidebar.

describePrimaryCompactLayouts ()

Retrieves metadata about the primary compact layout for each of the specified object types. Information returned is limited to 100 objects.

Syntax

```
DescribeCompactLayout[] primaryCompactLayouts =
connection.describePrimaryCompactLayouts(string[] sObjectType)
```

Usage

Use this call to retrieve information about the primary compact layout for the given object types. This call returns metadata about a given primary compact layout. For more information about compact layouts, see the Salesforce Help.

Sample Code—Java

```
public void testDescribePrimaryCompactLayoutsSample() {
    try {
        String[] objectsToDescribe = new String[] {"Account", "Lead"};
        DescribeCompactLayout[] primaryCompactLayouts =
connection.describePrimaryCompactLayouts(objectsToDescribe);

        for (int i = 0; i < primaryCompactLayouts.length; i++) {
            DescribeCompactLayout cLayout = primaryCompactLayouts[i];
            System.out.println(" There is a compact layout with name: " + cLayout.getName());

            // Write the objectType
            System.out.println(" This compact layout is the primary compact layout for: " +
```

```

cLayout.getObjectType());

    DescribeLayoutItem[] fieldItems = cLayout.getFieldItems();
    System.out.println(" There are " + fieldItems.length + " fields in this compact
layout.");

    // Write field items
    for (int j = 0; j < fieldItems.length; j++) {
        System.out.println(j + " This compact layout has a field with name: " +
fieldItems[j].getLabel());
    }

    DescribeLayoutItem[] imageItems = cLayout.getImageItems();
    System.out.println(" There are " + imageItems.length + " image fields in this
compact layout.");

    // Write the image items
    for (int j = 0; j < imageItems.length; j++) {
        System.out.println(j + " This compact layout has an image field with name: " +
imageItems[j].getLabel());
    }

    DescribeLayoutButton[] actions = cLayout.getActions();
    System.out.println(" There are " + actions.length + " buttons in this compact
layout.");

    // Write the action buttons
    for (int j = 0; j < actions.length; j++) {
        System.out.println(j + " This compact layout has a button with name: " +
actions[j].getLabel());
    }
}

} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

Arguments

Name	Type	Description
sObjectTypes	string[]	An array of one or more objects. The specified values must be valid objects for your organization.

Response

[DescribeCompactLayout](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

describeQuickActions ()



Retrieves details about specified actions.

Syntax

```
DescribeQuickActionResult[] = connection.describeQuickActions(string[] quickActionNames);
```

Usage

Use the `describeQuickActions ()` call to retrieve details for specified actions. In API version 28.0, the `describeQuickActions ()` call takes the action name in the form of `ParentEntity.ActionName`. In API version 29.0 and greater, it takes the action name in the form of `ContextEntity.ActionName`. Returns an array of `DescribeQuickActionResult`. For example, you can first call `describeAvailableQuickActions ()` for a list of actions available for a specified context and then use `describeQuickActions ()` to obtain details about specific actions.

-  **Note:** In API version 46.0 and later, the `apiName` for a global quick action can include the prefix `Global.` in a `describeQuickActions ()` request body. The request body also accepts global quick action API names without the prefix.
-  **Note:** The `describeQuickActions ()` call doesn't return results for actions created in your org if the Apex class was installed as part of a managed package.

Sample—Java

This sample retrieves and displays publisher action details for a create action on the Account object.

```
public void example() throws Exception {
    DescribeQuickActionResult[] result =
        conn.describeQuickActions(new String[]
            { "Account.QuickCreateContact", "Account.QuickCreateTask" });
    for(DescribeQuickActionResult r : result) {
        assert r != null;
        DescribeQuickActionDefaultValue [] describeQuickActionDefaultValues =
            r.getDefaultValues();
        for(DescribeQuickActionDefaultValue defaultValue : describeQuickActionDefaultValues)
        {
            System.out.println("Target Object Field: " + defaultValue.getField() );
            System.out.println("Target Object Field's default Value: " +
                defaultValue.getDefaultValue );
        }

        System.out.println("Action name: " + r.getName());
        System.out.println("Action label: " + r.getLabel());
        System.out.println("ParentOrContext object: " + r.getSourceSubjectType());
        System.out.println("Target object: " + r.getTargetSubjectType());
        System.out.println("Target object record type: " + r.getTargetRecordTypeId());
        System.out.println("Relationship field: " + r.getTargetParentField());
        System.out.println("Quick action type: " + r.getType());
        System.out.println("VF page name for custom actions: " +
```

```

    r.getVisualforcePageName();
    System.out.println("Icon name: " + r.getIconName());
    System.out.println("Icon URL: " + r.getIconUrl());
    System.out.println("Mini icon URL: " + r.getMiniIconUrl());
    assert r.getLayout() != null;
    System.out.println("Height of VF page for custom actions: " + r.getHeight());
    System.out.println("Width of VF page for custom actions: " + r.getWidth());
}
}

```

Arguments

Name	Type	Description
quickActions	string[]	An array of quick actions to be retrieved.

Response

[DescribeQuickActionResult](#)

DescribeQuickActionResult

The `describeQuickActions()` call returns an array of `DescribeQuickActionResult` objects. Each `DescribeQuickActionResult` object represents a quick action for a specified object.

Name	Type	Description
actionEnumOrId	string	The unique ID for the action. If the action doesn't have an ID, its API name is used. This field is available in API version 35.0 and later.
canvasApplicationName	string	The name of your Canvas application, if you use it.
colors	DescribeColor []	Array of color information. Each color is associated with a theme. This field is available in API version 29.0 and later.
defaultValues	DescribeQuickActionDefaultValue []	The action's default values.
height	int	The height in pixels of the action pane.
iconName	string	Name of icon used for the action. If a custom icon is not used, this value will not be set.
iconUrl	string	URL of icon used for the action. This icon URL corresponds to the 32x32 icon used for the current Salesforce theme, introduced in Spring '10, or the custom icon, if there is one.

Name	Type	Description
<code>icons</code>	DescribeIcon []	<p>Array of icons. Each icon is associated with a theme.</p> <p>If no custom icon was associated with the quick action and the quick action creates a specific object, the icons will correspond to the icons used for the created object. For example, if the quick action creates an Account, the icon array will contain the icons used for Account.</p> <p>If a custom icon was associated with the quick action, the array will contain that custom icon.</p> <p>This field is available in API version 29.0 and later. API version 32.0 and later returns different icons than in earlier API versions.</p>
<code>label</code>	string	Label of the action.
<code>layout</code>	DescribeLayoutSection	The section of the layout where the action resides.
<code>lightningComponentBundleId</code>	ID	<p>If <code>type</code> is <code>LightningComponent</code>, the ID of the Lightning component bundle called by the action.</p> <p>This field is available in API version 38.0 and later.</p>
<code>lightningComponentBundleName</code>	string	<p>If <code>type</code> is <code>LightningComponent</code>, the name of the Lightning component bundle called by the action.</p> <p>This field is available in API version 38.0 and later.</p>
<code>miniIconUrl</code>	string	The icon's URL. This icon URL corresponds to the 16x16 icon used for the current Salesforce theme, introduced in Spring '10, or the custom icon, if there is one.
<code>name</code>	string	Name of the action.
<code>contextObjectType</code>	string	The object used for the action. Named <code>sourceObjectType</code> in version 29.0 and earlier.
<code>showQuickActionVfHeader</code>	boolean	Whether or not the Visualforce quick action header and footer should be shown. If set to <code>false</code> , then both the header containing the quick action title and the footer containing the Save and Cancel buttons aren't displayed.
<code>targetParentField</code>	string	The parent object type of the action. Links the target object to the parent object. For example, use Account if the target object is Contact and the parent object is Account.
<code>targetRecordTypeId</code>	ID	The record type of the target record.
<code>targetObjectType</code>	string	The action's target object type.
<code>type</code>	string	<p>The action's type. Valid values are:</p> <ul style="list-style-type: none"> • Canvas • Create

Name	Type	Description
		<ul style="list-style-type: none"> • <code>Flow</code> (This value is available as a Beta in API version 41.0 and later) • <code>LightningComponent</code> (This value is available in API version 38.0 and later.) • <code>LogACall</code> • <code>Post</code> • <code>SendEmail</code> (This value is available in API version 31.0 and later.) • <code>SocialPost</code> • <code>Update</code> • <code>VisualforcePage</code>
<code>visualforcePageName</code>	string	If <code>type</code> is <code>Visualforce</code> , the page name of the associated page for the action.
<code>visualforcePageUrl</code>	string	If <code>type</code> is <code>Visualforce</code> , the URL of the associated page for the action.
<code>width</code>	int	If a custom action is created, this is the width in pixels of the action pane.

DescribeQuickActionDefaultValue

Represents the default values of fields to use in default layouts.

Name	Type	Description
<code>defaultValue</code>	string	The value of the auto-populated default action.
<code>field</code>	string	The field name of the action.

DescribeLayoutSection

Represents a section of a [DescribeLayout](#) and consists of one or more columns and one or more rows (an array of [DescribeLayoutRow](#)).

Name	Type	Description
<code>columns</code>	int	Number of columns in this <code>DescribeLayoutSection</code> .
<code>heading</code>	string	Heading text (label) for this <code>DescribeLayoutSection</code> .
<code>layoutRows</code>	<code>DescribeLayoutRow[]</code>	Array of one or more DescribeLayoutRow objects.
<code>parentLayoutId</code>	ID	The ID of the layout upon which this <code>DescribeLayoutSection</code> resides. This field is available in API version 35.0 and later.
<code>rows</code>	int	Number of rows in this <code>DescribeLayoutSection</code> .

Name	Type	Description
<code>tabOrder</code>	string	Indicates the tab order for the fields in the section in the edit view. Valid values are: <ul style="list-style-type: none"> • <code>LeftToRight</code> • <code>TopToBottom</code> This field is available in API version 31.0 and later.
<code>useCollapsibleSection</code>	boolean	Indicates whether this <code>DescribeLayoutSection</code> is a collapsible section, also known as a “twistie” (<code>true</code>), or not (<code>false</code>).
<code>useHeading</code>	boolean	Indicates whether to display the <code>heading</code> (<code>true</code>) or not (<code>false</code>).

DescribeLayoutRow

Represents a row in a [DescribeLayoutSection](#). A `DescribeLayoutRow` consists of one or more [DescribeLayoutItem](#) objects. For each `DescribeLayoutRow`, a `DescribeLayoutItem` refers either to a specific field or to an “empty” `DescribeLayoutItem` (a `DescribeLayoutItem` that contains no [DescribeLayoutComponent](#) objects). An empty `DescribeLayoutItem` can be returned when a given `DescribeLayoutRow` is sparse (for example, containing more fields on the right column than on the left column). Where there are gaps in the layout, an empty `DescribeLayoutItem` is returned as a placeholder.

Name	Type	Description
<code>layoutItems</code>	DescribeLayoutItem []	Refers to either a specific field or to an empty <code>LayoutItem</code> (a <code>LayoutItem</code> that contains no DescribeLayoutComponent objects).
<code>numItems</code>	int	Number of <code>layoutItems</code> . This information is redundant but, due to a bug in a popular SOAP toolkit, was required to avoid serialization problems.

DescribeLayoutItem

Represents an individual item in a [DescribeLayoutRow](#). A `DescribeLayoutItem` consists of a set of components ([DescribeLayoutComponent](#)), each of which is either a field or a separator. For most fields on a layout, there is only one component per layout item. However, in a display-only view, the `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 `DescribeLayoutItems`.

Name	Type	Description
<code>editable</code>	boolean	Indicates whether this <code>DescribeLayoutItem</code> can be edited (<code>true</code>) or not (<code>false</code>). This field is available in API version 30.0 and below. It was replaced by the <code>editableForNew</code> and <code>editableForUpdate</code> fields in API version 31.0.
<code>editableForNew</code>	boolean	Indicates whether a new <code>DescribeLayoutItem</code> can be edited when creating a new record (<code>true</code>) or not (<code>false</code>). This field is available in API version 31.0 and later.

Name	Type	Description
<code>editableForUpdate</code>	boolean	Indicates whether an existing DescribeLayoutItem can be edited when editing a record (<code>true</code>) or not (<code>false</code>). This field is available in API version 31.0 and later.
<code>label</code>	string	Label text for this DescribeLayoutItem.
<code>layoutComponents</code>	DescribeLayoutComponent[]	DescribeLayoutComponent for this DescribeLayoutItem.
<code>placeholder</code>	boolean	Indicates whether this DescribeLayoutItem is a placeholder (<code>true</code>) or not (<code>false</code>). If <code>true</code> , then this DescribeLayoutItem is blank.
<code>required</code>	boolean	Indicates whether this DescribeLayoutItem is required (<code>true</code>) or not (<code>false</code>). This is useful to know if, for example, you wanted to render required fields in a contrasting color (such as red).

DescribeLayoutComponent

Represents the smallest unit in a layout—a field or a separator. To reference a field for display, a client application uses the following notation to reference a field in the [describeSObjects\(\)](#) call: `LayoutComponent.fieldName`.

In API version 31.0 and later, DescribeLayoutComponent is extended with FieldLayoutComponent if both the [DescribeLayoutComponent](#) value is `Field`, and the field being described is either the compound field `Address` or the compound field `Person Name`.

Name	Type	Description
<code>displayLines</code>	int	The number of vertical lines displayed for a field in the edit view. Applies to <code>textarea</code> and multi-select picklist fields.
<code>tabOrder</code>	int	Indicates the tab order for the item in the row.
<code>type</code>	LayoutComponentType	The LayoutComponentType for this LayoutComponent.
<code>value</code>	string	Value of this LayoutComponent. The name of the field if the LayoutComponentType value is <code>Field</code> . The API name of the canvas app if the LayoutComponentType value is <code>Canvas</code> .

LayoutComponentType

Represents the type for a [DescribeLayoutComponent](#). Contains one of these values:

- `AnalyticsCloud`—A CRM Analytics dashboard on the page layout. Available in API version 34.0 and later.
- `Canvas`—A canvas component on the page layout. This layout component type is available in API version 31.0 and later.
- `CustomLink`—A custom link on the page layout.
- `EmptySpace`—A blank space on the page layout.
- `ExpandedLookup`—An Expanded Lookup component in the Mobile Cards section of the page layout.
- `Field`—Field name. A mapping to the `name` field on the [describeSObjectResult](#).
- `ReportChart`—A report chart on the page layout.
- `SControl`—Reserved for future use.

- `Separator`—Separator character, such as a semicolon (;) or slash (/).
- `VisualforcePage`—A Visualforce component on the page layout.

describeSearchScopeOrder ()

Retrieves an ordered list of the objects in a user's default global search scope.

Syntax

```
DescribeSearchScopeOrderResult[] describeSearchScopeOrderResults =  
connection.describeSearchScopeOrder ();
```

Usage

Use `describeSearchScopeOrder ()` to retrieve an ordered list of objects in the default global search scope of a logged-in user. Global search keeps track of which objects the user interacts with and how often and arranges the search results accordingly. Objects used most frequently appear at the top of the list. The returned list reflects the object order in the user's default search scope, including any pinned objects on the user's search results page. This call is useful if you want to implement a custom search results page using the optimized global search scope.

 **Note:** You must enable Chatter to enable global search.

Sample Code—Java

This sample shows how to retrieve the global search scope for a user and then iteratively display the name of each object in the scope.

```
public void describeSearchScopeOrderSample () {  
    try {  
        //Get the order of objects in search smart scope for the logged-in user  
        DescribeSearchScopeOrderResult[] describeSearchScopeOrderResults =  
            connection.describeSearchScopeOrder ();  
        //Iterate through the results and display the name of each object  
        for (int i = 0; i < describeSearchScopeOrderResults.length; i++) {  
            System.out.println (describeSearchScopeOrderResults [i].getName ());  
        }  
    }  
    catch (ConnectionException ce) {  
        ce.printStackTrace ();  
    }  
}
```

Arguments

None.

Response

An array of `DescribeSearchScopeOrderResult` objects

Fault

[UnexpectedErrorFault](#)

SEE ALSO:

[API Call Basics](#)

DescribeSearchScopeOrderResult

The `describeSearchScopeOrder()` call returns an array of `DescribeSearchScopeOrderResult` objects. Each `DescribeSearchScopeOrderResult` object represents an object in the user's global search scope. The list reflects the order of the objects in the user's scope, including any pinned objects. The `DescribeSearchScopeOrderResult` object has the following properties.

Name	Type	Description
<code>keyPrefix</code>	string	Three-character prefix code in the object ID. Object IDs are prefixed with three-character codes that specify the type of the object. For example, Account objects have a prefix of 001 and Opportunity objects have a prefix of 006. Note that a key prefix can sometimes be shared by multiple objects so it does not always uniquely identify an object.
<code>name</code>	string	Name of the object. English only.

`describeSearchLayouts()`

Retrieves the search result layout configuration for one or more objects.

Syntax

```
DescribeSearchLayoutResult[] = binding.describeSearchLayouts(string sObjectType[]);
```

Usage

Use `describeSearchLayouts()` to retrieve search layout information for one or more objects. This is handy when you want to create a custom search results page with the same layout settings as in Salesforce.

Sample

This sample shows how to retrieve the search result layout information for a list of objects.

```
public void describeSearchLayoutSample(String[] sObjectTypes) {
    try {
        // Get the search layout of Account and Group
        DescribeSearchLayoutResult[] searchLayoutResults =
connection.describeSearchLayouts(sObjectTypes);
        // Iterate through the results and display the label of each column
```

```

        for (int i = 0; i < sObjectTypes.length; i += 1) {
            String sObjectType = sObjectTypes[i];
            DescribeSearchLayoutResult result = searchLayoutResults[i];
            System.out.println("Top label for search results for " + sObjectType + "
is " + result.getLabel() + " and should display " + result.getLimitRows() + " rows");
            System.out.println("Column labels for search results for " + sObjectType
+ " are: ");
            for (DescribeColumn column : result.getSearchColumns()) {
                System.out.println(column.getLabel());
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}

```

Arguments

Name	Type	Description
sObjectType	string[]	The list of objects you want to obtain search result layout configuration for. For example, if the object is a person account, specify Account, or if it is a person contact, specify Contact. The specified values must be valid objects in your organization. For a complete list of standard objects, see Standard Objects .

Response

[DescribeSearchLayoutResult](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

DescribeSearchLayoutResult

The [describeSearchLayouts\(\)](#) on page 265 call returns an array of `DescribeSearchLayoutResult` objects. Each `DescribeSearchLayoutResult` object represents the search layout configuration for each object queried for. The `DescribeSearchLayoutResult` object has the following properties.

Name	Type	Description
label	string	The browser title used for the search results page.
limitRows	int	The maximum number of rows to be displayed in the first page of search results. This number can be changed by the administrator.

Name	Type	Description
searchColumns	DescribeColumn on page 266[]	The columns associated with the search results for this object.


DescribeColumn

Represents the columns in the search layout configuration for each `DescribeSearchLayoutResult` object returned by the `describeSearchLayouts()` on page 265 call.

Name	Type	Description
field	string	Field reference in relation to the object it belongs to. For example, "Lead.Phone."
format	string	Field data format. For example, "date". This value can be null.
label	string	Display text for this field in the user interface. For example, "Company Phone" or just "Phone."
name	string	Field name. Use this in your SOQL query or code. For example, "Name."

describeSObject()

Describes metadata (field list and object properties) for the specified object.

 **Note:** `describeSObjects()` supersedes `describeSObject()`. Use `describeSObjects()` instead of `describeSObject()`.

Syntax

```
DescribeSObjectResult = connection.describeSObject(string sObjectType);
```

Usage

Use `describeSObject()` to obtain metadata for a given object. You can first call `describeGlobal()` to retrieve a list of all objects for your organization, then iterate through the list and use `describeSObject()` to obtain metadata about individual objects.

Your client application must be logged in with sufficient access rights to retrieve metadata about your organization's data. For more information, see [Factors that Affect Data Access](#).

Sample Code—Java

This sample calls `describeSObject()` to perform describes on the Account sObject. It retrieves some properties of the sObject describe result, such as the sObject name, label, and fields. It then iterates through the fields and gets the field properties. For picklist

fields, it gets the picklist values and for reference fields, it gets the referenced object names. The sample writes the retrieved sObject and field properties to the console.

```
public void describeObjectSample() {
    try {
        // Make the describe call
        DescribeSObjectResult describeSObjectResult =
            connection.describeSObject("Account");

        // Get sObject metadata
        if (describeSObjectResult != null) {
            System.out.println("sObject name: " +
                describeSObjectResult.getName());
            if (describeSObjectResult.isCreateable())
                System.out.println("Createable");
        }

        // Get the fields
        Field[] fields = describeSObjectResult.getFields();
        System.out.println("Has " + fields.length + " fields");

        // Iterate through each field and gets its properties
        for (int i = 0; i < fields.length; i++) {
            Field field = fields[i];
            System.out.println("Field name: " + field.getName());
            System.out.println("Field label: " + field.getLabel());

            // If this is a picklist field, show the picklist values
            if (field.getType().equals(FieldType.picklist)) {
                PicklistEntry[] picklistValues =
                    field.getPicklistValues();
                if (picklistValues != null) {
                    System.out.println("Picklist values: ");
                    for (int j = 0; j < picklistValues.length; j++) {
                        if (picklistValues[j].getLabel() != null) {
                            System.out.println("\tItem: " +
                                picklistValues[j].getLabel()
                            );
                        }
                    }
                }
            }

            // If a reference field, show what it references
            if (field.getType().equals(FieldType.reference)) {
                System.out.println("Field references the " +
                    "following objects:");
                String[] referenceTos = field.getReferenceTo();
                for (int j = 0; j < referenceTos.length; j++) {
                    System.out.println("\t" + referenceTos[j]);
                }
            }
        }
    }
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
```

```

    }
}

```

Sample Code—C#

This sample calls `describeSObject()` to perform describes on the Account sObject. It retrieves some properties of the sObject describe result, such as the sObject name, label, and fields. It then iterates through the fields and gets the field properties. For picklist fields, it gets the picklist values and for reference fields, it gets the referenced object names. The sample writes the retrieved sObject and field properties to the console.

```

public void describeSObjectSample() {
    try {
        // Make the describe call
        DescribeSObjectResult describeSObjectResult =
            binding.describeSObject("Account");

        // Get sObject metadata
        if (describeSObjectResult != null) {
            Console.WriteLine("sObject name: " +
                describeSObjectResult.name);
            if (describeSObjectResult.createable)
                Console.WriteLine("Createable");
        }

        // Get the fields
        Field[] fields = describeSObjectResult.fields;
        Console.WriteLine("Has " + fields.Length + " fields");

        // Iterate through each field and gets its properties
        for (int i = 0; i < fields.Length; i++) {
            Field field = fields[i];
            Console.WriteLine("Field name: " + field.name);
            Console.WriteLine("Field label: " + field.label);

            // If this is a picklist field, show the picklist values
            if (field.type.Equals(fieldType.picklist)) {
                PicklistEntry[] picklistValues =
                    field.picklistValues;
                if (picklistValues != null) {
                    Console.WriteLine("Picklist values: ");
                    for (int j = 0; j < picklistValues.Length; j++) {
                        if (picklistValues[j].label != null) {
                            Console.WriteLine("\tItem: " +
                                picklistValues[j].label);
                        }
                    }
                }
            }
        }

        // If a reference field, show what it references
        if (field.type.Equals(fieldType.reference)) {
            Console.WriteLine("Field references the " +
                "following objects:");
            String[] referenceTos = field.referenceTo;

```

```

        for (int j = 0; j < referenceTos.Length; j++) {
            Console.WriteLine("\t" + referenceTos[j]);
        }
    }
}
}
} catch (SoapException e) {
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
}
}

```

Arguments

Name	Type	Description
sObjectType	string	Object. The specified value must be a valid object for your organization. For a complete list of objects, see Standard Objects .

Response

[DescribeSObjectResult](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[describeSObjects\(\)](#)

[describeGlobal\(\)](#)


[API Call Basics](#)

[Using the Partner WSDL](#)

https://developer.salesforce.com/page/Sample_SOAP_Messages

describeSObjectResult

The [describeSObject\(\)](#) call returns a DescribeSObjectResult object.

 **Note:** [describeSObjects\(\)](#) supersedes [describeSObject\(\)](#). Use [describeSObjects\(\)](#) instead of [describeSObject\(\)](#).

describeObjects ()

An array-based version of [describeObject \(\)](#); describes metadata (field list and object properties) for the specified object or array of objects.

 **Note:** Use this call instead of [describeObject \(\)](#).

Syntax

```
DescribeSObjectResult [] = connection.describeSObjects(string sObjectType[] );
```

Usage

Use [describeSObjects \(\)](#) to obtain metadata for a given object or array of objects. 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 \(\)](#) call is limited to a maximum of 100 objects returned.

Your client application must be logged in with sufficient access rights to retrieve metadata about your organization's data. For more information, see [Factors that Affect Data Access](#).

In organizations where person accounts are enabled, this call shows [Accounts](#) as not createable if the profile does not have access to any business account record types.

Sample Code—Java

This sample calls [describeSObjects \(\)](#) to perform describes on account, contact, and lead. It iterates through the sObject describe results, gets the properties and fields for each sObject in the result, and writes them to the console. For picklist fields, it writes the picklist values. For reference fields, it writes the referenced object names.

```
public void describeSObjectsSample ()
{
    try {
        // Call describeSObjectResults and pass it an array with
        // the names of the objects to describe.
        DescribeSObjectResult[] describeSObjectResults =
            connection.describeSObjects(
                new String[] { "account", "contact", "lead" });

        // Iterate through the list of describe sObject results
        for (int i=0;i < describeSObjectResults.length; i++)
        {
            DescribeSObjectResult desObj = describeSObjectResults[i];
            // Get the name of the sObject
            String objectName = desObj.getName();
            System.out.println("sObject name: " + objectName);

            // For each described sObject, get the fields
            Field[] fields = desObj.getFields();

            // Get some other properties
            if (desObj.getActivateable()) System.out.println("\tActivateable");
        }
    }
}
```

```

// Iterate through the fields to get properties for each field
for(int j=0;j < fields.length; j++)
{
    Field field = fields[j];
    System.out.println("\tField: " + field.getName());
    System.out.println("\t\tLabel: " + field.getLabel());
    if (field.isCustom())
        System.out.println("\t\tThis is a custom field.");
    System.out.println("\t\tType: " + field.getType());
    if (field.getLength() > 0)
        System.out.println("\t\tLength: " + field.getLength());
    if (field.getPrecision() > 0)
        System.out.println("\t\tPrecision: " + field.getPrecision());

    // Determine whether this is a picklist field
    if (field.getType() == FieldType.picklist)
    {
        // Determine whether there are picklist values
        PicklistEntry[] picklistValues = field.getPicklistValues();
        if (picklistValues != null && picklistValues[0] != null)
        {
            System.out.println("\t\tPicklist values = ");
            for (int k = 0; k < picklistValues.length; k++)
            {
                System.out.println("\t\t\tItem: " + picklistValues[k].getLabel());
            }
        }
    }

    // Determine whether this is a reference field
    if (field.getType() == FieldType.reference)
    {
        // Determine whether this field refers to another object
        String[] referenceTos = field.getReferenceTo();
        if (referenceTos != null && referenceTos[0] != null)
        {
            System.out.println("\t\tField references the following objects:");
            for (int k = 0; k < referenceTos.length; k++)
            {
                System.out.println("\t\t\t" + referenceTos[k]);
            }
        }
    }
}
} catch(ConnectionException ce) {
    ce.printStackTrace();
}
}

```

Sample Code—C#

This sample calls `describeObjects()` to perform describes on account, contact, and lead. It iterates through the `sObject` describe results, gets the properties and fields for each `sObject` in the result, and writes them to the console. For picklist fields, it writes the picklist values. For reference fields, it writes the referenced object names.

```
public void describeObjectsSample()
{
    try
    {
        // Call describeObjectResults and pass it an array with
        // the names of the objects to describe.
        DescribeSObjectResult[] describeSObjectResults =
            binding.describeObjects(
                new string[] { "account", "contact", "lead" });

        // Iterate through the list of describe sObject results
        foreach (DescribeSObjectResult describeSObjectResult in describeSObjectResults)
        {
            // Get the name of the sObject
            String objectName = describeSObjectResult.name;
            Console.WriteLine("sObject name: " + objectName);

            // For each described sObject, get the fields
            Field[] fields = describeSObjectResult.fields;

            // Get some other properties
            if (describeSObjectResult.activateable) Console.WriteLine("\tActivateable");

            // Iterate through the fields to get properties for each field
            foreach (Field field in fields)
            {
                Console.WriteLine("\tField: " + field.name);
                Console.WriteLine("\t\tLabel: " + field.label);
                if (field.custom)
                    Console.WriteLine("\t\tThis is a custom field.");
                Console.WriteLine("\t\tType: " + field.type);
                if (field.length > 0)
                    Console.WriteLine("\t\tLength: " + field.length);
                if (field.precision > 0)
                    Console.WriteLine("\t\tPrecision: " + field.precision);

                // Determine whether this is a picklist field
                if (field.type == fieldType.picklist)
                {
                    // Determine whether there are picklist values
                    PicklistEntry[] picklistValues = field.picklistValues;
                    if (picklistValues != null && picklistValues[0] != null)
                    {
                        Console.WriteLine("\t\tPicklist values = ");
                        for (int j = 0; j < picklistValues.Length; j++)
                        {
                            Console.WriteLine("\t\t\tItem: " + picklistValues[j].label);
                        }
                    }
                }
            }
        }
    }
}
```

```

    }
}

// Determine whether this is a reference field
if (field.type == fieldType.reference)
{
    // Determine whether this field refers to another object
    string[] referenceTos = field.referenceTo;
    if (referenceTos != null && referenceTos[0] != null)
    {
        Console.WriteLine("\t\tField references the following objects:");
        for (int j = 0; j < referenceTos.Length; j++)
        {
            Console.WriteLine("\t\t\t" + referenceTos[j]);
        }
    }
}
}
}

catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " + e.Message
        + "\n" + e.StackTrace);
}
}
}

```

Arguments

The `describeSObjects()` call takes in an array of `sObjects`.

Name	Type	Description
<code>sObjectType</code>	string	Object. The specified value must be a valid object for your organization. For a complete list of objects, see Standard Objects .

Response

[DescribeSObjectResult](#)

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:


[describeSObject\(\)](#)

[describeGlobal\(\)](#)

[API Call Basics](#)

[Using the Partner WSDL](#)

DescribeObjectResult

 **Important:** Where possible, we changed noninclusive terms to align with our company value of Equality. Because changing terms in our code can break current implementations, we maintained this object's name.


The `describeSObjects()` call returns an array of `DescribeObjectResult` objects. Each object has the following properties:

Name	Type	Description
<code>actionOverrides</code>	ActionOverride []	An array of action overrides. Action overrides replace the URLs specified in the <code>urlDetail</code> , <code>urlEdit</code> and <code>urlNew</code> fields. This field is available in API version 32.0 and later.
<code>activateable</code>	boolean	Reserved for future use.
<code>actionOverrides</code>	ActionOverride []	An array of action overrides. Action overrides replace the URLs specified in the <code>urlDetail</code> , <code>urlEdit</code> and <code>urlNew</code> fields. This field is available in API version 32.0 and later.
<code>associateEntityType</code>	string	If the object is associated with a parent object, the type of association it has to its parent, such as <code>History</code> . Otherwise, its value is <code>null</code> . Available in API version 50.0 and later.
<code>associateParentEntity</code>	string	If the object is associated with a parent object, the parent object it's associated with. Otherwise, its value is <code>null</code> . Available in API version 50.0 and later.
<code>childRelationships</code>	ChildRelationship []	An array of child relationships, which is the name of the <code>sObject</code> that has a foreign key to the <code>sObject</code> being described.
<code>compactLayoutable</code>	boolean	Indicates that the object can be used in describeCompactLayouts() .
<code>createable</code>	boolean	Indicates whether the object can be created via the <code>create()</code> call (<code>true</code>) or not (<code>false</code>).
<code>custom</code>	boolean	Indicates whether the object is a custom object (<code>true</code>) or not (<code>false</code>).
<code>customSetting</code>	boolean	Indicates whether the object is a custom setting object (<code>true</code>) or not (<code>false</code>).
<code>dataTranslationEnabled</code>	boolean	Indicates whether data translation is enabled for the object (<code>true</code>) or not (<code>false</code>). Available in API version 49.0 and later.
<code>deepCloneable</code>	boolean	Reserved for future use.

Name	Type	Description
<code>defaultImplementation</code>	string	Reserved for future use.
<code>deletable</code>	boolean	Indicates whether the object can be deleted via the <code>delete()</code> call (<code>true</code>) or not (<code>false</code>).
<code>deprecatedAndHidden</code>	boolean	Reserved for future use.
<code>extendedBy</code>	string	Reserved for future use.
<code>extendsInterfaces</code>	string	Reserved for future use.
<code>feedEnabled</code>	boolean	Indicates whether Chatter feeds are enabled for the object (<code>true</code>) or not (<code>false</code>). This property is available in API version 19.0 and later.
<code>fields</code>	Field[]	Array of fields associated with the object. The mechanism for retrieving information from this list varies among development tools.
<code>implementedBy</code>	string	Reserved for future use.
<code>implementsInterfaces</code>	string	Reserved for future use.
<code>isInterface</code>	boolean	Reserved for future use.
<code>keyPrefix</code>	string	<p>Three-character prefix code in the object ID. Object IDs are prefixed with three-character codes that specify the type of the object. For example, Account objects have a prefix of 001 and Opportunity objects have a prefix of 006. Note that a key prefix can sometimes be shared by multiple objects so it does not always uniquely identify an object.</p> <p>Use the value of this field to determine the object type of a parent in those cases where the child may have more than one object type as parent (polymorphic). For example, you may need to obtain the <code>keyPrefix</code> value for the parent of a Task or Event.</p>
<code>label</code>	string	Label text for a tab or field renamed in the user interface, if applicable, or the object name, if not. For example, an organization representing a medical vertical might rename <code>Account</code> to <code>Patient</code> . Tabs and fields can be renamed in the Salesforce user interface. See the Salesforce online help for more information.
<code>labelPlural</code>	string	Label text for an object that represents the plural version of an object name, for example, "Accounts."
<code>layoutable</code>	boolean	Indicates whether the object supports the <code>describeLayout()</code> call (<code>true</code>) or not (<code>false</code>).
<code>mergeable</code>	boolean	Indicates whether the object can be merged with other objects of its type (<code>true</code>) or not (<code>false</code>). <code>true</code> for leads, contacts, and accounts.
<code>mruEnabled</code>	boolean	Indicates whether Most Recently Used (MRU) list functionality is enabled for the object (<code>true</code>) or not (<code>false</code>).
<code>name</code>	string	Name of the object. This is the same string that was passed in as the <code>sObjectType</code> parameter.

Name	Type	Description
namedLayoutInfos	NamedLayoutInfo[]	The specific named layouts that are available for the objects other than the default layout.
networkScopeFieldName	string	The API name of the <code>networkScopeField</code> that scopes the entity to an Experience Cloud site. For most entities, the value of this property is <code>null</code> .
queryable	boolean	Indicates whether the object can be queried via the <code>query ()</code> call (<code>true</code>) or not (<code>false</code>).
recordTypeInfos	[]	An array of the record types supported by this object. The user need not have access to all the returned record types to see them here.
replicateable	boolean	Indicates whether the object can be replicated via the <code>getUpdated ()</code> and <code>getDeleted ()</code> calls (<code>true</code>) or not (<code>false</code>).
retrieveable	boolean	Indicates whether the object can be retrieved via the <code>retrieve ()</code> call (<code>true</code>) or not (<code>false</code>).
searchable	boolean	Indicates whether the object can be searched via the <code>search ()</code> call (<code>true</code>) or not (<code>false</code>).
searchLayoutable	boolean	Indicates whether search layout information can be retrieved via the <code>describeSearchLayouts ()</code> call (<code>true</code>) or not (<code>false</code>).
supportedScopes	ScopelInfo	The list of supported scopes for the object. For example, Account might have supported scopes of "All Accounts", "My Accounts", and "My Team's Accounts".
triggerable	boolean	Indicates whether the object supports Apex triggers.
undeletable	boolean	Indicates whether an object can be undeleted using the <code>undelete ()</code> call (<code>true</code>) or not (<code>false</code>).
updateable	boolean	Indicates whether the object can be updated via the <code>update ()</code> call (<code>true</code>) or not (<code>false</code>).
urlDetail	string	URL to the read-only detail page for this object. Compare with <code>urlEdit</code> , which is read-write. Client applications can use this URL to redirect to, or access, the Salesforce user interface for standard and custom objects. To provide flexibility and allow for future enhancements, returned <code>urlDetail</code> values are dynamic. To ensure that client applications are forward compatible, it is recommended that they use this capability where possible. Note that, for objects for which a stable URL is not available, this field is returned empty.
urlEdit	string	URL to the edit page for this object. For example, the <code>urlEdit</code> field for the Account object returns <code>https://yourInstance.salesforce.com/{ID}/e</code> . Substituting the <code>{ID}</code> field for the current object ID will return the edit page for that specific account in the Salesforce user interface. Compare with <code>urlDetail</code> , which is read-only. Client applications can use this URL to redirect to, or access, the Salesforce user interface for standard and custom objects. To provide flexibility and allow for future enhancements, returned <code>urlDetail</code> values are dynamic. To ensure that client applications are

Name	Type	Description
		forward compatible, it is recommended that they use this capability where possible. Note that, for objects for which a stable URL is not available, this field is returned empty.
<code>urlNew</code>	string	URL to the new/create page for this object. Client applications can use this URL to redirect to, or access, the Salesforce user interface for standard and custom objects. To provide flexibility and allow for future enhancements, returned <code>urlNew</code> values are dynamic. To ensure that client applications are forward compatible, it is recommended that they use this capability where possible. Note that, for objects for which a stable URL is not available, this field is returned empty.

 **Note:** The properties with a Boolean value indicate whether certain API calls can be used for an object. However, other factors, such as permissions, also affect whether such operations can be performed on the object.

ActionOverride

ActionOverride provides details about an action that replaces the default action pages for an object. For example, an object could be configured to replace the new/create page with a custom page. This type is available in API version 32.0 and later.

Name	Type	Description
<code>formFactor</code>	string	Represents the environment to which the action override applies. For example, a <code>Large</code> value in this field represents the Lightning Experience desktop environment, and is valid for Lightning pages and Lightning components. A <code>Small</code> value represents the Salesforce mobile app on a phone or tablet. This field is available in API version 37.0 and later.
<code>isAvailableInTouch</code>	boolean	Indicates whether the action override is available in the Salesforce mobile app (<code>true</code>) or not (<code>false</code>).
<code>name</code>	string	The name of the action that overrides the default action. For example, if the new/create page was overridden with a custom action, the name might be "New".
<code>pageId</code>	reference	The ID of the page for the action override.
<code>url</code>	string	The URL of the item being used for the action override, such as a Visualforce page. Returns as <code>null</code> for Lightning page overrides.

ChildRelationship


The name of the sObject that has a foreign key to the sObject being described.


Name	Type	Description
<code>cascadeDelete</code>	boolean	Indicates whether the child object is deleted when the parent object is deleted (<code>true</code>) or not (<code>false</code>).
<code>childSObject</code>	string	The name of the object on which there is a foreign key back to the parent <code>sObject</code> .
<code>deprecatedAndHidden</code>	boolean	Reserved for future use.
<code>field</code>	string	The name of the field that has a foreign key back to the parent <code>sObject</code> .
<code>junctionIdListNames</code>	String[]	The names of the lists of junction IDs associated with an object. Each ID represents an object that has a relationship with the associated object. For more information on <code>JunctionIdList</code> fields, see Field Types .
<code>junctionReferenceTo</code>	String[]	A collection of object names that the polymorphic keys in the <code>junctionIdListNames</code> property can reference. You can query these object names.
<code>relationshipName</code>	string	The name of the relationship, usually the plural of the value in <code>childSObject</code> .
<code>restrictedDelete</code>	boolean	Indicates whether the parent object can't be deleted because it is referenced by a child object (<code>true</code>) or not (<code>false</code>).

Field

In the `DescribeSObjectResult`, the `fields` property contains an array of `Field` objects. Each field represents a field in an API object. The array contains only the fields that the user can view, as defined by the user's field-level security settings.

Name	Type	Description
<code>autonumber</code>	boolean	Indicates whether this field is an autonumber field (<code>true</code>) or not (<code>false</code>). Analogous to a SQL <code>IDENTITY</code> type, autonumber fields are read only, non-createable text fields with a maximum length of 30 characters. Autonumber fields are read-only fields used to provide a unique ID that is independent of the internal object ID (such as a purchase order number or invoice number). Autonumber fields are configured entirely in the Salesforce user interface. The API provides access to this attribute so that client applications can determine whether a given field is an autonumber field.
<code>byteLength</code>	int	For variable-length fields (including binary fields), the maximum size of the field, in bytes.
<code>calculated</code>	boolean	Indicates whether the field is a custom formula field (<code>true</code>) or not (<code>false</code>). Note that custom formula fields are always read-only.
<code>caseSensitive</code>	boolean	Indicates whether the field is case sensitive (<code>true</code>) or not (<code>false</code>).

Name	Type	Description
controllerName	string	The name of the field that controls the values of this picklist. It only applies if <code>type</code> is <code>PICKLIST</code> or <code>MULTIPICKLIST</code> and <code>dependentPicklist</code> is <code>true</code> . The mapping of controlling field to dependent field is stored in the <code>validFor</code> attribute of each <code>PicklistEntry</code> for this picklist.
createable	boolean	Indicates whether the field can be created (<code>true</code>) or not (<code>false</code>). If <code>true</code> , then this field value can be set in a <code>create()</code> call.
custom	boolean	Indicates whether the field is a custom field (<code>true</code>) or not (<code>false</code>).
dataTranslationEnabled	boolean	Indicates whether data translation is enabled for the field (<code>true</code>) or not (<code>false</code>). Available in API version 49.0 and later.
defaultedOnCreate	boolean	Indicates whether this field is defaulted when created (<code>true</code>) or not (<code>false</code>). If <code>true</code> , then Salesforce implicitly assigns a value for this field when the object is created, even if a value for this field is not passed in on the <code>create()</code> call. For example, in the <code>Opportunity</code> object, the <code>Probability</code> field has this attribute because its value is derived from the <code>Stage</code> field. Similarly, the <code>Owner</code> has this attribute on most objects because its value is derived from the current user (if the <code>Owner</code> field is not specified).
defaultValueFormula	string	The default value specified for this field if the formula is not used. If no value has been specified, this field is not returned.
dependentPicklist	boolean	Indicates whether a picklist is a dependent picklist (<code>true</code>) where available values depend on the chosen values from a controlling field, or not (<code>false</code>). See About Dependent Picklists .
deprecatedAndHidden	boolean	Reserved for future use.
digits	int	For fields of type integer. Maximum number of digits. The API returns an error if an integer value exceeds the number of digits.
displayLocationInDecimal	boolean	Indicates how the geolocation values of a <code>Location</code> custom field appears in the user interface. If <code>true</code> , the geolocation values appear in decimal notation. If <code>false</code> , the geolocation values appear as degrees, minutes, and seconds.
encrypted	boolean	 Note: This page is about Shield Platform Encryption, not Classic Encryption. What's the difference? Indicates whether this field is encrypted. This value only appears in the results of a <code>describeObjects()</code> call when it is <code>true</code> ; otherwise, it is omitted from the results. This field is available in API version 31.0 and later.
extraTypeInfo	string	If the field is a <code>textarea</code> field type, indicates if the text area is plain text (<code>plaintextarea</code>) or rich text (<code>richtextarea</code>). If the field is a <code>url</code> field type, if this value is <code>imageurl</code> , the URL references an image file. Available on standard fields on standard objects only, for example, <code>Account.imageUrl</code> , <code>Contact.imageUrl</code> , and so on. If the field is a <code>reference</code> field type, indicates the type of external object relationship. Available on external objects only.

Name	Type	Description
		<ul style="list-style-type: none"> • <code>null</code>—lookup relationship • <code>externallookup</code>—external lookup relationship • <code>indirectlookup</code>—indirect lookup relationship
<code>filterable</code>	boolean	Indicates whether the field is filterable (<code>true</code>) or not (<code>false</code>). If <code>true</code> , then this field can be specified in the <code>WHERE</code> clause of a query string in a <code>query()</code> call.
<code>filteredLookupInfo</code>	FilteredLookupInfo	If the field is a <code>reference</code> field type with a lookup filter, <code>filteredLookupInfo</code> contains the lookup filter information for the field. If there is no lookup filter, or the filter is inactive, this field is <code>null</code> . This field is available in API version 31.0 and later.
<code>formula</code>	string	The formula specified for this field. If no formula is specified for this field, it is not returned.
<code>groupable</code>	boolean	Indicates whether the field can be included in the <code>GROUP BY</code> clause of a SOQL query (<code>true</code>) or not (<code>false</code>). See <code>GROUP BY</code> in the Salesforce SOQL and SOSL Reference Guide . Available in API version 18.0 and later.
<code>highScaleNumber</code>	boolean	Indicates whether the field stores numbers to 8 decimal places regardless of what's specified in the field details (<code>true</code>) or not (<code>false</code>). Used to handle currencies for products that cost fractions of a cent, in large quantities. If high-scale unit pricing isn't enabled in your organization, this field isn't returned. Available in API version 33.0 and later.
<code>htmlFormatted</code>	boolean	Indicates whether a field such as a hyperlink custom formula field has been formatted for HTML and should be encoded for display in HTML (<code>true</code>) or not (<code>false</code>). Also indicates whether a field is a custom formula field that has an <code>IMAGE</code> text function.
<code>idLookup</code>	boolean	Indicates whether the field can be used to specify a record in an <code>upsert()</code> call (<code>true</code>) or not (<code>false</code>).
<code>inlineHelpText</code>	string	The text that displays in the field-level help hover text for this field.  Note: This property is not returned unless at least one field on the object contains a value. When at least one field has field-level help, all fields on the object list the property with either the field-level help value or null for fields that have blank field-level help.
<code>label</code>	string	Text label that is displayed next to the field in the Salesforce user interface. This label can be localized.
<code>length</code>	int	Returns the maximum size of the field in Unicode characters (not bytes) or 255, whichever is less. The maximum value returned by the <code>getLength()</code> property is 255. Available in API version 49.0 and later.
<code>mask</code>	string	Reserved for future use.
<code>maskType</code>	string	Reserved for future use.

Name	Type	Description
name	string	Field name used in API calls, such as <code>create()</code> , <code>delete()</code> , and <code>query()</code> .
nameField	boolean	Indicates whether this field is a name field (<code>true</code>) or not (<code>false</code>). Used to identify the name field for standard objects (such as <code>AccountName</code> for an Account object) and custom objects. Limited to one per object, except where <code>FirstName</code> and <code>LastName</code> fields are used (such as in the Contact object). If a compound name is present, for example the <code>Name</code> field on a person account, <code>nameField</code> is set to <code>true</code> for that record. If no compound name is present, <code>FirstName</code> and <code>LastName</code> have this field set to <code>true</code> .
namePointing	boolean	Indicates whether the field's value is the Name of the parent of this object (<code>true</code>) or not (<code>false</code>). Used for objects whose parents may be more than one type of object, for example a task may have an account or a contact as a parent.
nillable	boolean	Indicates whether the field is nillable (<code>true</code>) or not (<code>false</code>). A nillable field can have empty content. A non-nillable field must have a value in order for the object to be created or saved.
permissionable	boolean	Indicates whether FieldPermissions can be specified for the field (<code>true</code>) or not (<code>false</code>).
picklistValues	PicklistEntry[]	Provides the list of valid values for the picklist. Specified only if <code>restrictedPicklist</code> is <code>true</code> .
polymorphicForeignKey	boolean	Indicates whether the foreign key includes multiple entity types (<code>true</code>) or not (<code>false</code>).
precision	int	For fields of type double. 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).
relationshipName	string	The name of the relationship, if this is a master-detail relationship field.
relationshipOrder	int	The type of relationship for a master-detail relationship field. Valid values are: <ul style="list-style-type: none"> • 0 if the field is the primary relationship • 1 if the field is the secondary relationship
referenceTargetField	string	Applies only to indirect lookup relationships on external objects. Name of the custom field on the parent standard or custom object whose values are matched against the values of the child external object's indirect lookup relationship field. This matching is done to determine which records are related to each other. This field is available in API version 32.0 and later.
referenceTo	string[]	For fields that refer to other objects, this array indicates the object types of the referenced objects.
restrictedPicklist	boolean	Indicates whether the field is a restricted picklist (<code>true</code>) or not (<code>false</code>).
scale	int	For fields of type double. Number of digits to the right of the decimal point. The API silently truncates any extra digits to the right of the decimal point, but it returns a fault response if the number has too many digits to the left of the decimal point.

Name	Type	Description
searchPrefilterable	boolean	Indicates whether a foreign key can be included in prefiltering (<code>true</code>) or not (<code>false</code>) when used in a SOSL WHERE clause. <i>Prefiltering</i> means to filter by a specific field value before executing the full search query. Available in API version 40.0 and later.
soapType	SOAPType	See SOAPType for a list of allowable values.
sortable	boolean	Indicates whether a query can sort on this field (<code>true</code>) or not (<code>false</code>).
type	FieldType	See FieldType for a list of allowable values.
unique	boolean	Indicates whether the value must be unique (<code>true</code>) or not (<code>false</code>).
updateable	boolean	Indicates one of the following: <ul style="list-style-type: none"> Whether the field is updateable, (<code>true</code>) or not (<code>false</code>). If <code>true</code>, then this field value can be set in an <code>update()</code> call. If the field is in a master-detail relationship on a custom object, indicates whether the child records can be reparented to different parent records (<code>true</code>), <code>false</code> otherwise.
writeRequiresMasterRead	boolean	This field only applies to master-detail relationships. Indicates whether a user requires read sharing access (<code>true</code>) or write sharing access (<code>false</code>) to the parent record to insert, update, and delete a child record. In both cases, a user also needs Create, Edit, and Delete object permissions for the child object.

FieldType

In the `Field` object associated with `DescribeObjectResult`, the `type` field can contain one of the following strings. For more information about field types, see [Field Types](#).

type Field Value	What the Field Object Contains
string	String values.
boolean	Boolean (<code>true</code> / <code>false</code>) values.
int	Integer values.
double	Double values.
date	Date values.
datetime	Date and time values.
base64	Base64-encoded arbitrary binary data (of type <code>base64Binary</code>). Used for Attachment , Document , and Scontrol objects.
ID	Primary key field for the object. For information on IDs, see Field Types .
reference	Cross-references to a different object. Analogous to a foreign key field in SQL.
currency	Currency values.

type Field Value	What the Field Object Contains
textarea	String that is displayed as a multiline text field.
percent	Percentage values.
phone	Phone numbers. Values can include alphabetic characters. Client applications are responsible for phone number formatting.
url	URL values. Client applications should commonly display these as hyperlinks. If <code>Field.extraTypeInfo</code> is <code>imageUrl</code> , the URL references an image, and can be displayed as an image instead.
email	Email addresses.
combobox	Comboboxes, which provide a set of enumerated values and allow the user to specify a value not in the list.
picklist	Single-select picklists, which provide a set of enumerated values from which only one value can be selected.
multipicklist	Multi-select picklists, which provide a set of enumerated values from which multiple values can be selected.
anyType	Values can be any of these types: <code>string</code> , <code>picklist</code> , <code>boolean</code> , <code>int</code> , <code>double</code> , <code>percent</code> , <code>ID</code> , <code>date</code> , <code>dateTime</code> , <code>url</code> , or <code>email</code> .
location	Geolocation values, including latitude and longitude, for custom geolocation fields on custom objects.

FilteredLookupInfo

In the `Field` object associated with the `DescribeObjectResult`, the `filteredLookupInfo` field contains information about the lookup filter associated with the field.

This subtype is available in API version 31.0 and later.

Name	Type	Description
<code>controllingFields</code>	<code>string[]</code>	Array of the field's controlling fields when the lookup filter is dependent on the source object.
<code>dependent</code>	<code>boolean</code>	Indicates whether the lookup filter is dependent upon the source object (<code>true</code>) or not (<code>false</code>).
<code>optionalFilter</code>	<code>boolean</code>	Indicates whether the lookup filter is optional (<code>true</code>) or not (<code>false</code>).

SOAPType

The `DescribeObjectResult` returns the `fields` property, which contains an array of fields whose value provides information about the object being described. One of those fields, `soapType`, contains one of the following string values. All of the values preceded by `xsd:` are XML schema primitive data types. For more information about the XML schema primitive data types, see the World Wide Web Consortium's publication *XML Schema Part 2: Data Types* at: <http://www.w3.org/TR/xmlschema-2/>.

Value	Description
<code>tns:ID</code>	Unique ID associated with an sObject. For information on IDs, see Field Types .
<code>xsd:anyType</code>	Can be ID, Boolean, double, integer, string, date, or dateTime.
<code>xsd:base64Binary</code>	Base 64-encoded binary data.
<code>xsd:boolean</code>	Boolean (<code>true</code> / <code>false</code>) values.
<code>xsd:date</code>	Date values.
<code>xsd:dateTime</code>	Date/time values.
<code>xsd:double</code>	Double values.
<code>xsd:int</code>	Integer values.
<code>xsd:string</code>	Character strings.

PicklistEntry

In the `Field` object associated with the `DescribeObjectResult`, the `picklistValues` field contains an array of `PicklistEntry` properties. Each `PicklistEntry` can contain any one of the following string values. For more information, see [Field Types](#).

Name	Type	Description
<code>active</code>	boolean	Indicates whether this item must be displayed (<code>true</code>) or not (<code>false</code>) in the drop-down list for the picklist field in the user interface.
<code>validFor</code>	byte[]	A set of bits where each bit indicates a controlling value for which this <code>PicklistEntry</code> is valid. See About Dependent Picklists .
<code>defaultValue</code>	boolean	Indicates whether this item is the default item (<code>true</code>) in the picklist or not (<code>false</code>). Only one item in a picklist can be designated as the default.
<code>label</code>	string	Display name of this item in the picklist.
<code>value</code>	string	Value of this item in the picklist.

About Dependent Picklists

A dependent picklist works in conjunction with a controlling field to filter its values. The value chosen in the controlling field affects the values available in the dependent picklist.

A dependent picklist can be any custom picklist or multi-select picklist field that displays available values based on the value selected in its corresponding controlling field. A controlling field can be any standard or custom picklist (with at least one and less than 200 values) or checkbox field whose values control the available values in one or more corresponding dependent fields.

In the following example, the controlling picklist `Beverage` has two values, which relate to the values of the dependent picklist `Beverage Variety`:

Beverage	Beverage Variety
Coffee	Decaffeinated

Beverage	Beverage Variety
	Regular
Tea	Chamomile
	Earl Grey
	English Breakfast

For each `PicklistEntry` that represents a value in a dependent picklist, the `validFor` attribute contains a set of bits. Each bit indicates a controlling field value for which the `PicklistEntry` is valid. Read the bits from left to right.

For more information on dependent picklists, see the “Dependent Picklists” topic in the Salesforce online help.

Sample Java Code for Dependent Picklists

```
public void dependentPicklistSample() {
    // inner class to decode a "validFor" bitset
    class Bitset {
        byte[] data;

        public Bitset(byte[] data) {
            this.data = data == null ? new byte[0] : data;
        }

        public boolean testBit(int n) {
            return (data[n >> 3] & (0x80 >> n % 8)) != 0;
        }

        public int size() {
            return data.length * 8;
        }
    }

    try {
        DescribeSObjectResult describeSObjectResult = connection.describeSObject("Case");
        Field[] fields = describeSObjectResult.getFields();
        // create a map of all fields for later lookup
        Map fieldMap = new HashMap();
        for (int i = 0; i < fields.length; i++) {
            fieldMap.put(fields[i].getName(), fields[i]);
        }
        for (int i = 0; i < fields.length; i++) {
            // check whether this is a dependent picklist
            if (fields[i].getDependentPicklist()) {
                // get the controller by name
                Field controller = (Field)fieldMap.get(fields[i].getControllerName());
                System.out.println("Field '" + fields[i].getLabel() + "' depends on '" +
                    controller.getLabel() + "'");
                PicklistEntry[] picklistValues = fields[i].getPicklistValues();
                for (int j = 0; j < picklistValues.length; j++) {
                    // for each PicklistEntry: list all controlling values for which it is valid
                }
            }
        }
    }
}
```



```

System.out.println("Item: " + picklistValues[j].getLabel() +
" is valid for: ");
Bitset validFor = new Bitset(picklistValues[j].getValidFor());
if (FieldType.picklist == controller.getType()) {
    // if the controller is a picklist, list all
    // controlling values for which this entry is valid
    for (int k = 0; k < validFor.size(); k++) {
        if (validFor.testBit(k)) {
            // if bit k is set, this entry is valid for the
            // for the controlling entry at index k
            System.out.println(controller.getPicklistValues()[k].getLabel());
        }
    }
} else if (FieldType._boolean == controller.getType()) {
    // the controller is a checkbox
    // if bit 1 is set this entry is valid if the controller is checked
    if (validFor.testBit(1)) {
        System.out.println(" checked");
    }
    // if bit 0 is set this entry is valid if the controller is not checked
    if (validFor.testBit(0)) {
        System.out.println(" unchecked");
    }
}
}
}
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

RecordTypeInfo

Base class for the old RecordTypeMapping object. This object contains all of the existing fields of RecordTypeMapping except layoutId and picklistForRecordType.

Name	Type	Description
available	boolean	Indicates whether this record type is available (<code>true</code>) or not (<code>false</code>). Availability is used to display a list of available record types to the user when they are creating a new record.
defaultRecordTypeMapping	boolean	Indicates whether this is the default record type mapping (<code>true</code>) or not (<code>false</code>).
developerName	string	Developer name of this record type. Available in API versions 43.0 and later.
master	boolean	Indicates whether this is the main record type (<code>true</code>) or not (<code>false</code>). The main record type is the default record type that's used when a record has no custom record type associated with it.
name	string	Name of this record type.

Name	Type	Description
recordTypeId	ID	ID of this record type.

NamedLayoutInfo

The name of the named layout for the object. Standard objects can have defined named layouts which are separate from the primary layout for both the profile and the record type. For more information on layout names, see [describeLayout\(\)](#).

Name	Type	Description
name	string	Name of this layout.

ScopeInfo

A scope for an object that can be used to filter object records. For example, Account may have a supported ScopeInfo of “mine” (with a UI label of “My accounts”) which filters only Account records for the current user.

Name	Type	Description
label	string	UI label for this scope.
name	string	Name of this scope.

describeSoftphoneLayout ()

Retrieves layout information for a Salesforce CRM Call Center Softphone.

Syntax

```
DescribeSoftphoneLayoutResult[] = connection.describeSoftphoneLayout ();
```

Usage

Use this call to obtain information about the layout of a Softphone. Use only in the context of Salesforce CRM Call Center; do not call directly from client programs.

Arguments

This call does not take any objects.

Response

The response is a DescribeSoftphoneLayoutResult object:

Name	Type	Description
callTypes	DescribeSoftphoneLayoutCallType[]	A set of attributes associated with each allowed call type. A call type may be Inbound, Outbound, or Internal.
id	ID	ID of layout. Note that layout objects are not exposed via the API.
name	string	Name of the call type: Inbound, Outbound, or Internal.

DescribeSoftphoneLayoutCallType

Each DescribeSoftphoneLayoutResult object contains one or more call types:

Name	Type	Description
infoFields	DescribeSoftphoneLayoutInfoField[]	A set of information field in the softphone layout.
name	string	Name of the layout.
screenPopOptions	DescribeSoftphoneScreenPopOption[]	Settings in the softphone layout that specify how to display screen pops when the details of calls match or don't match existing records. This field is available in API version 18.0 and later.
screenPopsOpenWithin	string	Setting in the softphone layout that specify whether to display screen pops in a new browser window or tab when the details of calls match or don't match existing records. This field is available in API version 18.0 and later.
sections	DescribeSoftphoneLayoutSection[]	A set of object names and the corresponding item name in the softphone layout. There is one section for each object in a call type.

DescribeSoftphoneLayoutInfoField

An information field in the softphone layout.

Name	Type	Description
name	string	The name of an information field in the softphone layout that does not correspond to a Salesforce object. For example, caller ID may be specified in an information field. Information fields hold static information about the call type.

DescribeSoftphoneLayoutSection

Each call type returned in a DescribeSoftphoneLayoutResult object contains one section for each call type. Each section contains object-item pairs:

Name	Type	Description
entityApiName	string	The name of an object in the Salesforce application that corresponds to an item displayed in the softphone layout, for example, a set of accounts or cases.
items	DescribeSoftphoneLayoutItem []	A set of softphone layout items.

DescribeSoftphoneLayoutItem

Each layout item corresponds to a record in Salesforce:

Name	Type	Description
itemApiName	string	The name of a record in the Salesforce application that corresponds to an item displayed in the softphone layout, for example, the Acme account.

DescribeSoftphoneScreenPopOption

Each call type returned in a `DescribeSoftphoneLayoutResult` object contains one `screenPopOptions` field for each call type. Each `screenPopOptions` field contains details about screen pop settings:

Name	Type	Description
matchType	string	Setting on a softphone layout to pop a screen for call details that match a single record, multiple records, or no records.
screenPopData	string	Setting on a softphone layout for a specific object or page to pop for a call's <code>matchType</code> . For example, pop a specified Visualforce page when the details of a call match a record.
screenPopType	picklist	Setting that specifies how to pop a screen for a call's <code>matchType</code> . For example, pop a detail page or don't pop any page when the details of a call match a record.

Sample Code—Java

This sample describes the soft phone layout and writes its properties to the console. It then gets the allowed call types. For each call type, it gets its information fields, layout sections, and the layout items in the layout sections. It writes these values to the console.

```
public void describeSoftphoneLayout () {
    try {
        DescribeSoftphoneLayoutResult result =
            connection.describeSoftphoneLayout ();
        System.out.println("ID of retrieved Softphone layout: " +
            result.getId());
        System.out.println("Name of retrieved Softphone layout: " +
            result.getName());
        System.out.println("\nContains following " +
            "Call Type Layouts\n");
        for (DescribeSoftphoneLayoutCallType type :
            result.getCallTypes()) {
```

```

System.out.println("Layout for " + type.getName() +
    " calls");
System.out.println("\tCall-related fields:");
for (DescribeSoftphoneLayoutInfoField field :
    type.getInfoFields()) {
    System.out.println("\t\t{" + field.getName());
}
System.out.println("\tDisplayed Objects:");
for (DescribeSoftphoneLayoutSection section :
    type.getSections()) {
    System.out.println("\t\tFor entity " +
        section.getEntityApiName() +
        " following records are displayed:");
};
for (DescribeSoftphoneLayoutItem item :
    section.getItems()) {
    System.out.println("\t\t\t" + item.getItemApiName());
}
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

Sample Code—C#

This sample describes the soft phone layout and writes its properties to the console. It then gets the allowed call types. For each call type, it gets its information fields, layout sections, and the layout items in the layout sections. It writes these values to the console.

```

/// Demonstrates how to retrieve the layout information
/// for a Salesforce CRM Call Center Softphone
public void DescribeSoftphoneLayoutSample()
{
    try
    {
        DescribeSoftphoneLayoutResult dsplResult = binding.describeSoftphoneLayout();

        // Display the ID and Name of the layout
        Console.WriteLine("ID of retrieved Softphone layout: {0}", dsplResult.id);
        Console.WriteLine("Name of retrieved Softphone layout: {0}", dsplResult.name);

        // Display the contents of each Call Type
        Console.WriteLine("\nContains following Call Type Layouts\n");
        foreach (DescribeSoftphoneLayoutCallType dsplCallType in dsplResult.callTypes)
        {
            Console.WriteLine("Layout for {0} calls", dsplCallType.name);

            // Display the call-related fields contained in the call type
            Console.WriteLine("\tCall-related fields:");
            foreach (DescribeSoftphoneLayoutInfoField dsplInfoField
                in dsplCallType.infoFields)
            {

```

```

        Console.WriteLine("\t\t{0}", dsplInfoField.name);
    }

    // Display the objects that are included in the layout
    Console.WriteLine("\tDisplayed Objects:");
    foreach (DescribeSoftphoneLayoutSection dsplSection
        in dsplCallType.sections)
    {
        Console.WriteLine("\t\tFor entity {0} following records are displayed:",
            dsplSection.entityApiName);
        foreach (DescribeSoftphoneLayoutItem dsplItem in dsplSection.items)
        {
            Console.WriteLine("\t\t\t{0}", dsplItem.itemApiName);
        }
    }
}
}
catch (SoapException e)
{
    Console.WriteLine(e.Message);
    Console.WriteLine(e.StackTrace);
    Console.WriteLine(e.InnerException);
}
}
}

```

describeSoqlListViews ()

Retrieves the SOQL query and other information about a list view.

Syntax

```
connection.describeSoqlListViews(DescribeSoqlListViewsRequest request);
```

Usage

Use the `describeSoqlListViews ()` call to retrieve information about a list view, including the ID, the columns, and the SOQL query. This call is useful if you want to use the SOQL that drives an existing list view in your custom application. This call is available in API version 32.0 and later.

Sample Code—Java

```

public void example() throws Exception {
    DescribeSoqlListViewsRequest request =
createDescribeSoqlListViewsRequest(listViewId, null);
    this.getClient().describeSoqlListViews(request);
}
}

```

Arguments

Name	Type	Description
request	DescribeSoqlListViewsRequest	The fully qualified name or the ID of the list view and the object with which the list view is associated.

Response

A [DescribeSoqlListViewResult](#) object that contains one or more [DescribeSoqlListView](#) objects.

Faults

[InvalidSObjectFault](#)

[UnexpectedErrorFault](#)

DescribeSoqlListView

Contains information about the specified list view, including the columns, sObject type, and SOQL query.

The [DescribeSoqlListView](#) object has the following properties:

Name	Type	Description
columns	ListViewColumn []	The columns that are returned by the list view query.
id	ID	The list view's fully qualified ID.
orderBy	ListViewOrderBy []	A list of fields to sort results by, the sort order, and the position to which null values should be sorted.
query	string	The fully composed SOQL query for the list view.
relatedEntityId	ID	The ID of the campaign, if a campaign scope was used.
scope	string	A filterScope to use for limiting the results.
scopeEntityId	ID	The ID of the queue or price book, if a queue or price book scope was used.
sObjectType	string	The object with which the list view is associated.
whereCondition	SoqlWhereCondition	Filter conditions on the list view. Filter conditions provide an additional level of control over which records get shown in the list view.

SEE ALSO:

https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_select_using_scope.htm

DescribeSoqlListViewParams

Use the [DescribeSoqlListViewParams](#) object with [describeSoqlListViews\(\)](#) to retrieve the SOQL from a list view.

The `DescribeSoqlListViewParams` object has the following properties:

Name	Type	Description
<code>developerNameOrId</code>	string	The list view's ID or fully qualified developer name.
<code>subjectType</code>	string	The API name of the sObject for the list view.

DescribeSoqlListViewResult

Contains one or more `DescribeSoqlListView` objects, each of which contains information about one or more list views, including the ID, sObject type, columns, and SOQL query of each.

The `DescribeSoqlListViewResult` object has the following properties:

Name	Type	Description
<code>describeSoqlListViews</code>	DescribeSoqlListView[]	Information about one or more list views, including the ID, sObject type, columns, and SOQL query of each.

DescribeSoqlListViewsRequest

Use the `DescribeSoqlListViewsRequest` object with `describeSoqlListViews()` to retrieve information about a list view.

The `DescribeSoqlListViewsRequest` object has the following properties:

Name	Type	Description
<code>listViewParams</code>	DescribeSoqlListViewParams[]	A list of parameters that specify the list view to describe.

EDITIONS

Available in: Salesforce Classic

Available in:

ListViewColumn

Contains metadata about a single list view column.

The `ListViewColumn` object is returned by the `describeSoqlListViews()` and `executeListView()` calls. It has the following properties:

Name	Type	Description
<code>ascendingLabel</code>	string	The localized type-specific label for sorting the column in ascending order. For example: "A-Z" for a text field, or "Low to High" for a numeric field. Set to null if the column isn't sortable.
<code>descendingLabel</code>	string	The localized type-specific label for sorting the column in ascending order. For example: "Z-A" for a text field, or "High to Low" for a numeric field. Set to null if the column is not sortable.
<code>fieldNameOrPath</code>	string	The field name or SOQL field path for the column.
<code>hidden</code>	boolean	If true, specifies that the column is not displayed, and is present only to support the display of other columns or other client-side logic.

Name	Type	Description
label	string	The localized display label for the column.
searchable	boolean	Whether the column is searchable.
selectListItem	string	The SOQL SELECT item for the column. The item might differ from the field name or path, due to display formatting (for example, toLabel for picklists).
sortDirection	orderByDirection	An enumerated value, one of the following if the column is sortable: <ul style="list-style-type: none"> ascending descending Set to null if the column is not sortable.
sortIndex	int	The zero-based index that indicates the column's position within a multilevel sort, or null if the records are not sorted by the column.
sortable	boolean	Whether the column is sortable, in which case it might be referenced in the ExecuteListView orderBy parameter.
type	FieldType	The column data type.

ListViewOrderBy

Use the `ListViewOrderBy` object with `executeListView()` to determine the order in which records are returned from a list view.

The `ListViewOrderBy` object is returned by the `describeSoqlListViews()` call, is an optional input to the `executeListView()` call, and has the following properties:

Name	Type	Description
fieldNameOrPath	string	The field name or SOQL path of the field on which to sort the records.
nullsPosition	orderByNullsPosition	An enumerated value that determines where nulls are sorted in the results: <ul style="list-style-type: none"> first last
sortDirection	orderByDirection	An enumerated value that determines the sort order of the results: <ul style="list-style-type: none"> ascending descending

SoqlWhereCondition

Contains information about SOQL filter conditions for a list view.

Each condition listed in `SoqlWhereCondition` represents a condition expression in a SOQL WHERE clause that compares a field value to a comparison value using a condition operator. Each condition contains the following properties.

Name	Type	Description
field	string	The object field used by the filter condition.
operator	sqlOperator	<p>The filter operation. Operations include:</p> <ul style="list-style-type: none"> • equals—Condition is true if the field value equals the specified value. String comparisons using the equals operator are case sensitive for unique case-sensitive fields and case insensitive for all other fields. • excludes—Condition is true for multi-select picklist fields if the selected field values are not in the list of condition values. • greaterThan—Condition is true if the field value is greater than the specified value. • greaterThanOrEqualTo—Condition is true if the field value is greater than or equal to the specified value. • in—Condition is true if the field value equals any specified value in the values list. • includes—Condition is true for multi-select picklist fields if the selected field values are in the list of condition values. • lessThan—Condition is true if the field value is less than the specified value. • lessThanOrEqualTo—Condition is true if the field value is less than or equal to the specified value. • like—Condition is true if the field value matches the specified value, using character matching logic described in Comparison Operators in the <i>SOQL and SOSL Reference</i>. • notEquals—Condition is true if the field value doesn't equal the specified value. • notIn—Condition is true if the field value doesn't equal any specified value in the values list. • notLike—Condition is true if the field value doesn't match the specified value using the character matching logic described in Comparison Operators in the <i>SOQL and SOSL Reference</i>. Available in API version 41.0 and later. • within—Condition is true if the field value location is within the value distance using a location-based comparison. For more information, see Location-Based SOQL Queries in the <i>SOQL and SOSL Reference</i>.
values	string[]	A list of one or more values used to compare with the field value using the <code>operator</code> comparison logic.

Evaluating SqlWhereConditions

In the SOAP API, Salesforce uses subclasses of `SqlWhereCondition` to represent different categories of conditions. Use your development language's type comparison functionality (such as Java's `instanceof` operator) to determine which subclass is used for a particular instance of `SqlWhereCondition`.

The `SqlConditionGroup` subclass represents a group of SOQL WHERE clause conditions and uses the following properties.

Name	Type	Description
conditions	condition[]	List of filter conditions. If the list view uses filter logic, each logical filter group is represented with a conditions list.
conjunction	soqlConjunction	A conjunction operation that describes the filter logic to use for multiple conditions in a logical filter group. Values include: <ul style="list-style-type: none"> and—All conditions must be true for the overall SoqlWhereCondition. or—One of the conditions must be true for the overall SoqlWhereCondition.

The `SoqlNotCondition` subclass represents a special use of the `like` operator. In API version 40.0 and earlier, when evaluating a `SoqlWhereCondition` that was created using a `not like` operator (displayed as **does not contain** in the UI), the operator value in the condition is `like`. Salesforce also uses the `SoqlNotCondition` subclass of `SoqlWhereCondition` to represent the complete condition. The following example uses Java's `instanceof` operator to determine whether a `not like` operation is specified.

```
if (resultSoqlWhereCondition instanceof SoqlNotCondition) {
    // condition is really NOT condition
    // if operator is "like", this condition really means "not like"
    ...
}
```

In API version 41.0 and later, the `notLike` operator is used instead of `SoqlNotCondition` and a `like` operator. The `notLike` operator is available only for list views. You can't use it in SOQL queries used in other Salesforce features.

describeTabs ()

Returns information about the Salesforce Classic standard and custom apps available to the logged-in user, as listed in the Lightning Platform app menu at the top of the page. An app is a set of tabs that works as a unit to provide application functionality. For example, two of the standard Salesforce apps are "Sales" and "Service."

Syntax

```
describeTabSetResult [] = connection.describeTabs ();
```

Usage

Use the `describeTabs ()` call to obtain information about the Salesforce Classic standard and custom apps to which the logged-in user has access. The `describeTabs ()` call returns the minimum required metadata that can be used to render apps in another user interface. Typically this call is used by partner applications to render Salesforce data in another user interface.

For each app, the call returns the app name, the URL of the logo, whether it's the currently selected application for the user, and details about the tabs included in that app.

Important: The `describeTabs ()` call returns information only about tabs that display in the Salesforce user interface for the logged-in user. If a user clicks the All Tabs (+) tab and hides some tabs from his Salesforce user interface, those user-hidden tabs aren't included in the set of tabs returned by `describeTabs ()`.

Use the `describeAllTabs ()` call to obtain information about all the tabs that are available to the logged-in user.

For each tab, the call returns the tab name, the primary `sObject` that's displayed on the tab, whether it's a custom tab, and the URL for viewing that tab. Note that the "All Tabs" tab and Lightning page tabs aren't included in the list of tabs.

Sample Code—Java

This sample calls `describeTabs()`, which returns an array of tab set results. Next, for each tab set result, which represents a Salesforce Classic app, it retrieves some of its properties and gets all the tabs for this app. It writes all retrieved properties to the console.

```
public void describeTabsSample() {
    try {
        // Describe tabs
        DescribeTabSetResult[] dtsrs = connection.describeTabs();
        System.out.println("There are " + dtsrs.length +
            " tab sets defined.");

        // For each tab set describe result, get some properties
        for (int i = 0; i < dtsrs.length; i++) {
            System.out.println("Tab Set " + (i + 1) + ":");
            DescribeTabSetResult dtsr = dtsrs[i];
            System.out.println("Label: " + dtsr.getLabel());
            System.out.println("\tLogo URL: " + dtsr.getLogoUrl());
            System.out.println("\tTab selected: " +
                dtsr.isSelected());

            // Describe the tabs for the tab set
            DescribeTab[] tabs = dtsr.getTabs();
            System.out.println("\tTabs defined: " + tabs.length);

            // Iterate through the returned tabs
            for (int j = 0; j < tabs.length; j++) {
                DescribeTab tab = tabs[j];
                System.out.println("\tTab " + (j + 1) + ":");
                System.out.println("\t\tName: " +
                    tab.getSObjectName());
                System.out.println("\t\tLabel: " + tab.getLabel());
                System.out.println("\t\tURL: " + tab.getUrl());
                DescribeColor[] tabColors = tab.getColors();
                // Iterate through tab colors as needed
                DescribeIcon[] tabIcons = tab.getIcons();
                // Iterate through tab icons as needed
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample calls `describeTabs()`, which returns an array of tab set results. Next, for each tab set result, which represents a Salesforce Classic app, it retrieves some of its properties and gets all the tabs for this app. It writes all retrieved properties to the console.

```
public void describeTabsSample() {
    try {
        // Describe tabs
        DescribeTabSetResult[] dtsrs = binding.describeTabs();
        Console.WriteLine("There are " + dtsrs.Length +
            " tab sets defined.");

        // For each tab set describe result, get some properties
        for (int i = 0; i < dtsrs.Length; i++) {
            Console.WriteLine("Tab Set " + (i + 1) + ":");
            DescribeTabSetResult dtsr = dtsrs[i];
            Console.WriteLine("Label: " + dtsr.label);
            Console.WriteLine("\tLogo URL: " + dtsr.logoUrl);
            Console.WriteLine("\tTab selected: " +
                dtsr.selected);

            // Describe the tabs for the tab set
            DescribeTab[] tabs = dtsr.tabs;
            Console.WriteLine("\tTabs defined: " + tabs.Length);

            // Iterate through the returned tabs
            for (int j = 0; j < tabs.Length; j++) {
                DescribeTab tab = tabs[j];
                Console.WriteLine("\tTab " + (j + 1) + ":");
                Console.WriteLine("\t\tName: " +
                    tab.subjectName);
                Console.WriteLine("\t\tLabel: " + tab.label);
                Console.WriteLine("\t\tURL: " + tab.url);
                DescribeColor[] tabColors = tab.colors;
                // Iterate through tab colors as needed
                DescribeIcon[] tabIcons = tab.icons;
                // Iterate through tab icons as needed
            }
        }
    } catch (SoapException e) {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

Arguments

None.

Response

[describeTabSetResult](#), [DescribeTab](#)

SEE ALSO:

[API Call Basics](#)

[Using the Partner WSDL](#)

[DescribeTab](#)

[describeTabSetResult](#)

describeTabSetResult

The `describeTabs()` call returns an array of `DescribeTabSetResult` objects for Salesforce Classic standard or custom apps and has these properties:

Name	Type	Description
<code>description</code>	string	The description for this standard or custom app.
<code>label</code>	string	The display label for this standard or custom app. This value changes when tabs are renamed in the Salesforce user interface. See Salesforce Help for more information.
<code>logoUrl</code>	string	A fully qualified URL to the logo image associated with the standard or custom app.
<code>namespace</code>	string	If this is a custom app, and a set of tabs in the custom app was installed as part of a managed package, the value of this attribute is the developer namespace prefix that the creator of the package chose when the Developer Edition organization was enabled to allow publishing a managed package. This attribute identifies elements of a Salesforce AppExchange package.
<code>selected</code>	boolean	If <code>true</code> , then this standard or custom app is the user's currently selected app.
<code>tabs</code>	DescribeTab	An array of tabs that are displayed for the specified standard app or custom app.

DescribeColor

`DescribeColor` contains color metadata information for a tab. The `describeTabs()` call returns an array of [DescribeTabSetResult](#) values. Each `DescribeTabSetResult` contains an array of [DescribeTab](#) values, and each `DescribeTab` contains an array of `DescribeColor` values.

Each `DescribeColor` is associated with a Salesforce user interface theme. For more information on themes, see [Identifying the Salesforce Style Your Users See](#) in the Visualforce Developer's Guide.

Color information can also be retrieved via the `describeTheme()` and `describeGlobalTheme()` calls. These calls return information on colors used for each object in your organization that can use theme icons and colors.

Name	Type	Description
<code>color</code>	string	The color described in web color RGB format—for example, "00FF00".

Name	Type	Description
context	string	The color context, which determines whether the color is the main color (or primary) for the tab.
theme	string	The associated theme. Possible values include: <ul style="list-style-type: none"> <i>theme2</i>—Theme used prior to Spring '10, called the "Salesforce Classic 2005 user interface theme" <i>theme3</i>—Theme introduced in Spring '10, called the "Salesforce Classic 2010 user interface theme" <i>theme4</i>—Theme introduced in Winter '14 for the mobile touchscreen version of Salesforce, and in Winter '16 for Lightning Experience <i>custom</i>—Theme associated with a custom icon

DescribeIcon

DescribeIcon contains icon metadata information for a tab. The `describeTabs()` call returns an array of `DescribeTabSetResult` values. Each `DescribeTabSetResult` contains an array of `DescribeTab` values, and each `DescribeTab` contains an array of `DescribeIcon` values.

Icon information can also be retrieved via the `describeTheme()` and `describeGlobalTheme()` calls. These calls return information on icons used for each object in your organization that can use theme icons and colors.

Name	Type	Description
contentType	string	The tab icon's content type, for example, "image/png."
height	int	The tab icon's height in pixels. If the icon content type is an SVG type, height and width values are not used.
theme	string	The associated theme. Possible values include: <ul style="list-style-type: none"> <i>theme2</i>—Theme used prior to Spring '10, called the "Salesforce Classic 2005 user interface theme" <i>theme3</i>—Theme introduced in Spring '10, called the "Salesforce Classic 2010 user interface theme" <i>theme4</i>—Theme introduced in Winter '14 for the mobile touchscreen version of Salesforce, and in Winter '16 for Lightning Experience <i>custom</i>—Theme associated with a custom icon
url	string	The fully qualified URL for this icon.
width	string	The tab icon's width in pixels. If the icon content type is an SVG type, height and width values are not used.

DescribeTab

The `describeTabs()` call returns a `describeTabSetResult` object, of which `DescribeTab` is a property:

Name	Type	Description
<code>colors</code>	DescribeColor []	Array of color information used for a tab. This field is available in API version 29.0 and later.
<code>custom</code>	boolean	<code>true</code> if this is a custom tab, <code>false</code> if this is a standard tab.
<code>iconUrl</code>	string	The URL for the main 32x32 pixel icon for a tab. This icon appears next to the heading at the top of most pages. This icon URL corresponds to the 32x32 icon used for the Salesforce Classic 2010 user interface theme.
<code>icons</code>	DescribeIcon []	Array of icon information used for a tab. This field is available in API version 29.0 and later.
<code>label</code>	string	The display label for this tab.
<code>miniIconUrl</code>	string	The URL for the 16x16 pixel icon that represents a tab. This icon appears in related lists and other locations. This icon URL corresponds to the 16x16 icon used for the current Salesforce theme, introduced in Spring '10.
<code>name</code>	string	The API name of the tab.
<code>subjectName</code>	string	The name of the <code>sObject</code> that is primarily displayed on this tab (for tabs that display a particular <code>sObject</code>). For a list of objects, see Standard Objects .
<code>url</code>	string	A fully qualified URL for viewing this tab.

SEE ALSO:

[DescribeColor](#)

[DescribeIcon](#)

describeTheme ()

Returns information about themes available to the current logged-in user.

Syntax

```
DescribeThemeResult = connection.describeTheme(string sObjectType[]);
```

Usage

Use `describeTheme()` to get current theme information for a given array of objects. Theme information consists of colors and icons for an object in Salesforce, used for a particular theme. For example, the `Merchandise__c` object might use the "computer32" icon and a primary tab color of red for the regular Salesforce application theme, and a different set of colors and icons for the mobile touchscreen version of Salesforce.

If you pass `null` instead of an array of objects, `describeTheme()` returns theme information for all objects in your organization that use theme colors and icons.

Your client application must be logged in with sufficient access rights to retrieve theme information about your organization's data. For more information, see [Factors that Affect Data Access](#).

`describeTheme()` is available in API version 29.0 and later.

Sample

This Java sample calls `describeTheme()` to retrieve theme information for Account and Contact, and then iterates over the retrieved theme information.

```
public static void describeThemeExample() {
    try {
        // Get current themes
        DescribeTheme themeResult = connection.describeTheme(
            new String[] { "Account", "Contact" });
        DescribeThemeItem[] themeItems = themeResult.getThemeItems();
        for (int i = 0; i < themeItems.length; i++) {
            DescribeThemeItem themeItem = themeItems[i];
            System.out.println("Theme information for object " + themeItem.getName());
            // Get color and icon info for each themeItem
            DescribeColor colors[] = themeItem.getColors();
            System.out.println("    Number of colors: " + colors.length);
            int k;
            for (k = 0; k < colors.length; k++) {
                DescribeColor color = colors[k];
                System.out.println("        For Color #" + k + ":");
                System.out.println("            Web RGB Color: " + color.getColor());
                System.out.println("            Context: " + color.getContext());
                System.out.println("            Theme: " + color.getTheme());
            }
            DescribeIcon icons[] = themeItem.getIcons();
            System.out.println("    Number of icons: " + icons.length);
            for (k = 0; k < icons.length; k++) {
                DescribeIcon icon = icons[k];
                System.out.println("        For Icon #" + k + ":");
                System.out.println("            ContentType: " + icon.getContentType());
                System.out.println("            Height: " + icon.getHeight());
                System.out.println("            Theme: " + icon.getTheme());
                System.out.println("            URL: " + icon.getUrl());
                System.out.println("            Width: " + icon.getWidth());
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Response

[DescribeThemeResult](#)

Faults

[UnexpectedErrorFault](#)

SEE ALSO:

[DescribeThemeResult](#)

[DescribeThemeItem](#)

[DescribeColor](#)

[DescribeIcon](#)

DescribeThemeResult

The [describeTheme\(\)](#) and [describeGlobalTheme\(\)](#) calls return [DescribeThemeResult](#), which contains an array of [DescribeThemeItem](#) values.

Name	Type	Description
themes	DescribeThemeItem []	Array of themes. Theme information is provided for each object in the organization that can use theme icons and colors.

DescribeThemeItem

The [describeTheme\(\)](#) and [describeGlobalTheme\(\)](#) calls return [DescribeThemeResult](#), which contains an array of [DescribeThemeItem](#) values. Each [DescribeThemeItem](#) contains an array of colors and icons used for themes, and the name of the object the theme information applies to.

Name	Type	Description
colors	DescribeColor []	Array of colors.
icons	DescribeIcon []	Array of icons.
name	string	Name of the object that the theme colors and icons are associated with.

CHAPTER 11 Utility Calls

This topic describes API calls that your client applications can invoke to obtain the system timestamp, user information, and change user passwords.

 **Note:** For a list of Apex-related calls, see [Apex-Related Calls](#), for a list of core calls, see [Core Calls](#), and for a list of describe calls, see [Describe Calls](#).

The following table lists the API utility calls described in this topic:

Task / Call	Description
<code>getServerTimestamp()</code>	Retrieves the current system timestamp from the API.
<code>changeOwnPassword()</code>	Allows users to change their own passwords.
<code>getUserInfo()</code>	Retrieves personal information for the user associated with the current session.
<code>match()</code>	Evaluates sObjects provided as an input for matches among Leads, using the matching rule specified in the input MatchOptions. This call can be used only with the Standard Matching Rule for Leads on Accounts.
<code>renderEmailTemplate()</code>	Replaces merge fields in text bodies of email templates with values from Salesforce records, even for polymorphic fields. The email template bodies and their corresponding <code>whoId</code> and <code>whatId</code> values are specified in the argument.
<code>resetPassword()</code>	Changes a user's password to a system-generated value.
<code>sendEmail()</code>	Immediately sends an email message.
<code>sendEmailMessage()</code>	Immediately sends up to 10 draft email messages.
<code>setPassword()</code>	Sets the specified user's password to the specified value.

Samples

The samples in this section are based on the enterprise WSDL file. They assume that you have already imported the WSDL file and created a connection. To learn how to do so, see the [Quick Start](#) tutorial.

`changeOwnPassword()`

Allows users to change their passwords from old values to new values that they specify.

Syntax

```
ChangeOwnPasswordResult changeOwnPasswordResult = connection.changeOwnPassword(string
oldPassword, string newPassword);
```

Usage

Use `changeOwnPassword()` to allow users to change their passwords to values that they specify. For example, a client application prompts a user to specify a different password, and then invokes `changeOwnPassword()` to change the user's password. Use `setPassword()` if you want to set a different user's password to a value you specify. Use `resetPassword()` if you want to reset a target user's password with a random value generated by the API.

Sample Code—Java

This sample accepts old password and new password parameters, which it uses in the `changeOwnPassword()` call to set the new password of the user.

```
public void doChangeOwnPassword(String oldPasswd, String newPasswd) {
    try {
        ChangeOwnPasswordResult result = connection.changeOwnPassword(oldPasswd, newPasswd);

        System.out.println("Your password was changed to "
            + newPasswd);
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample accepts old password and new password parameters, which it uses in the `changeOwnPassword()` call to set the new password of the user.

```
public void doChangeOwnPassword(String oldPasswd, String newPasswd)
{
    try
    {
        ChangeOwnPasswordResult result = binding.changeOwnPassword(oldPasswd, newPasswd);
        Console.WriteLine("Your password was changed to "
            + newPasswd);
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

Arguments

Name	Type	Description
oldPassword	string	The user's previous password that is being replaced.
newPassword	string	The user's new password.

Response

ChangeOwnPasswordResult

Fault

InvalidOldPasswordFault

InvalidNewPasswordFault

[UnexpectedErrorFault](#)

SEE ALSO:

[resetPassword\(\)](#)

[Utility Calls](#)

[setPassword\(\)](#)

getServerTimestamp ()

Retrieves the current system timestamp (Coordinated Universal Time (UTC) time zone) from the API.


Syntax

```
GetServerTimestampResult timestamp = connection.getServerTimestamp();
```

Usage

Use [getServerTimestamp \(\)](#) to obtain the current system timestamp from the API. You might do this if, for example, you need to use the exact timestamp for timing or data synchronization purposes. When you [create \(\)](#) or [update \(\)](#) an object, the API uses the system timestamp to update the `CreatedDate` and `LastModifiedDate` fields, respectively, in the object.

The [getServerTimestamp \(\)](#) call always returns the timestamp in Coordinated Universal Time (UTC) time zone. However, your local system might automatically display the results in your local time based on your time zone settings.

 **Note:** Development tools differ in the way that they handle time data. Some development tools report the local time, while others report only the Coordinated Universal Time (UTC) time zone. To determine how your development tool handles time values, refer to its documentation.

Sample Code—Java

This sample gets the server time and writes it to the console in the user's local time zone.

```
public void doGetServerTimestamp() {
    try {
        GetServerTimestampResult result = connection.getServerTimestamp();
        Calendar serverTime = result.getTimestamp();
        System.out.println("Server time is: "
            + serverTime.getTime().toString());
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample gets the server time and writes it to the console in the user's local time zone.

```
public void doGetServerTimestamp()
{
    try
    {
        GetServerTimestampResult result =
            binding.getServerTimestamp();
        DateTime serverTime = result.timestamp;
        Console.WriteLine("Server time is: " +
            serverTime.ToLocalTime().ToString());
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

Arguments

None.

Response

[getServerTimestampResult](#)

Fault

[UnexpectedErrorFault](#)

SEE ALSO:

[Utility Calls](#)

getServerTimestampResult

The `getServerTimestamp()` call returns a `GetServerTimestampResult` object, which has the following properties:

Name	Type	Description
timestamp	dateTime	System timestamp of the API when the <code>getServerTimestamp()</code> call was executed.

getUserInfo()

Retrieves personal information for the user associated with the current session.

Syntax

```
getUserInfoResult result = connection.getUserInfo();
```

Usage

Use `getUserInfo()` to obtain personal information about the currently logged-in user. This convenience API call retrieves and aggregates common profile information that your client application can use for display purposes, performing currency calculations, and so on.

The `getUserInfo()` call applies only to the username under which your client application has logged in. To retrieve additional personal information not found in the `getUserInfoResult` object, you can call `retrieve()` on the `User` object and pass in the `userID` returned by this call. To retrieve personal information about other users, you could call `retrieve()` (if you know their user ID) or `query()` on the `User` object.

Sample Code—Java

This sample calls `getUserInfo()` and writes information about the current user to the console.

```
public void doGetUserInfo() {
    try {
        GetUserInfoResult result = connection.getUserInfo();
        System.out.println("\nUser Information");
        System.out.println("\tFull name: " + result.getUserFullName());
        System.out.println("\tEmail: " + result.getUserEmail());
        System.out.println("\tLocale: " + result.getUserLocale());
        System.out.println("\tTimezone: " + result.getUserTimeZone());
        System.out.println("\tCurrency symbol: " + result.getCurrencySymbol());
        System.out.println("\tOrganization is multi-currency: " +
            result.isOrganizationMultiCurrency());
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample calls `getUserInfo()` and writes information about the current user to the console.

```
public void doGetUserInfo()
{
    try
    {
        GetUserInfoResult result = binding.getUserInfo();
        Console.WriteLine("\nUser Information");
        Console.WriteLine("\tFull name: " + result.userFullName);
        Console.WriteLine("\tEmail: " + result.userEmail);
        Console.WriteLine("\tLocale: " + result.userLocale);
        Console.WriteLine("\tTimezone: " + result.userTimeZone);
        Console.WriteLine("\tCurrency symbol: " + result.currencySymbol);
        Console.WriteLine("\tOrganization is multi-currency: " +
            result.organizationMultiCurrency);
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

Arguments

None.

Response

[getUserInfoResult](#)

Fault

[UnexpectedErrorFault](#)

SEE ALSO:

[Utility Calls](#)

getUserInfoResult

The `getUserInfo()` call returns a `GetUserInfoResult` object.

Name	Type	Description
<code>accessibilityMode</code>	<code>boolean</code>	Available in API version 7.0 and later. Indicates whether user interface modifications for the visually impaired are on (<code>true</code>) or off (<code>false</code>). The modifications facilitate the use of screen readers such as JAWS.

Name	Type	Description
<code>chatterExternal</code>	boolean	Type of user license assigned to the Profile associated with the user. Indicates whether a user is part of the org or external. Available in API 40.0 and later.
<code>currencySymbol</code>	string	Currency symbol to use for displaying currency values. Applicable only when <code>organizationMultiCurrency</code> is <code>false</code> .
<code>organizationId</code>	ID	ID of the organization. Allows third-party tools to uniquely identify individual organizations in Salesforce, which is useful for retrieving billing or organization-wide setup information.
<code>organizationMultiCurrency</code>	boolean	Indicates whether the user's organization uses multiple currencies (<code>true</code>) or not (<code>false</code>).
<code>organizationName</code>	string	Name of the user's organization or company.
<code>orgDefaultCurrencyIsoCode</code>	string	Default currency ISO code. Applicable only when <code>organizationMultiCurrency</code> is <code>false</code> . When the logged-in user creates any objects that have a currency ISO code, the API uses this currency ISO code if it is not explicitly specified in the <code>create()</code> call.
<code>profileID</code>	ID	ID of the profile associated with the role currently assigned to the user.
<code>roleID</code>	ID	Role ID of the role currently assigned to the user.
<code>sessionSecondsValid</code>	int	The number of seconds before the session expires, starting from the last update time. Available in API version 21.0 and later.
<code>userDefaultCurrencyIsoCode</code>	string	Default currency ISO code. Applicable only when <code>organizationMultiCurrency</code> is <code>true</code> . When the logged-in user creates any objects that have a currency ISO code, the API uses this currency ISO code if it is not explicitly specified in the <code>create()</code> call.
<code>userEmail</code>	string	User's email address.
<code>userFullName</code>	string	User's full name.
<code>userID</code>	ID	User ID.
<code>userLanguage</code>	string	User's language, which controls the language for labels displayed in an application. String is 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 country, then the string also has an underscore (<code>_</code>) and another ISO country code, for example "US" or "UK. For example, the string for the United States is "en_US", and the string for French Canadian is "fr_CA." For a list of the languages that Salesforce supports, see the Salesforce online help topic "What languages does Salesforce support?"
<code>userLocale</code>	string	User's locale, which controls the formatting of dates and choice of symbols for currency. The first two characters are always an ISO language code, for example "fr" or "en." If the value is further qualified by country, then the string also has

Name	Type	Description
		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."
userName	string	User's login name.
userTimeZone	string	User's time zone.
userType	string	Type of user license assigned to the Profile associated with the user.
userUiSkin	string	Available in API version 7.0 and later. Possible values are: <ul style="list-style-type: none"> • <code>theme3</code>—If the user is using the Salesforce Classic 2010 user interface theme, also known as the <i>Aloha</i> interface • <code>theme2</code>—If the user is using the Salesforce Classic 2005 user interface theme • <code>theme1</code>—If the user is using the oldest user interface theme (obsolete) In the online app, this look and feel setting is configurable from Setup by entering <i>User Interface</i> in the <code>Quick Find</code> box, then selecting User Interface . See User Interface Themes .

match ()

Evaluates sObjects provided as an input for matches among Leads, using the matching rule specified in the input MatchOptions. This call can be used only with the Standard Matching Rule for Leads on Accounts.

This operation is available in API versions 42.0 and later, in **Professional**, **Enterprise**, **Performance**, and **Unlimited** Editions with **Pardot Pro** or **Pardot Ultimate** Edition.

Syntax

```
MatchResult[] callResults = connection.match(SObject[] inputSObjectArray, MatchOptions matchOptions);
```

Arguments

Name	Type	Description
inputSObjectArray	Array of sObject	A list of sObjects to evaluate for matches.
matchOptions	MatchOptions	Options, such as the match rule, used during the match operation.

Response





























[MatchResult](#)













MatchOptions

Represents a type to be used with a match operation. It describes options to be used during the match operation. This type can be used only with the Standard Matching Rule for Leads on Accounts.

This type is available in API versions 42.0 and later, in **Professional**, **Enterprise**, **Performance**, and **Unlimited** Editions with **Pardot Pro** or **Pardot Ultimate** Edition.

Fields

Field	Details	
Fields	string	
MatchEngine	string	           
Rule	string	              

Field	Details	
		     
SubjectType	string	     

renderEmailTemplate()

Replaces merge fields in text bodies of email templates with values from Salesforce records, even for polymorphic fields. The email template bodies and their corresponding `whoId` and `whatId` values are specified in the argument.

Syntax

```
RenderEmailTemplateResult = connection.renderEmailTemplate(RenderEmailTemplateRequest[] renderRequests);
```

Usage

The `renderEmailTemplate()` call is equivalent to rendering merge fields when sending an email with a custom template through the `sendEmail()` call.

The `renderEmailTemplate()` call can take up to 10 `RenderEmailTemplateRequest` elements in its array argument, and each `RenderEmailTemplateRequest` can contain up to 10 template bodies. Each request is independent from the other requests in the array—an error in one request doesn't affect the other requests. Similarly, an error in one template body doesn't cause an error in other text bodies within the same request.

The `renderEmailTemplate()` call substitutes a merge field with the value of either the `whatId` or `whoId` in `RenderEmailTemplateRequest`:

- If the merge field references a non-human object, it's replaced with the corresponding value of `whatId`. For example, if a merge field references an account or opportunity, the `whatId` value is substituted.
- If the merge field references a human object, it's replaced with the corresponding value of `whoId`. For example, if a merge field references a contact, lead, or user, the `whoId` value is substituted.

The `whatId` and `whoId` field values of `RenderEmailTemplateRequest` are validated for each request. If the `whatId` doesn't reference a valid what ID (a non-human object), or the `whoId` doesn't reference a valid who ID (a human object), an error is set for the request.

Sample Code—Java

In this sample, the `renderEmailTemplate()` call substitutes all contact merge fields with the value from the specified `whoId` argument. Similarly, the call substitutes the opportunity merge field (`{!Opportunity.Name}`) with the specified `whatId` value. The second template body in this sample has an incorrect merge field (`{!Contact.SNARF}`), which causes an error on the second template. However, the entire template rendering request is successful.

```
public void renderTemplates(String whoId, String whatId)
    throws ConnectionException, RemoteException, MalformedURLException {
    // Array of three template bodies.
    // The second template body generates an error.
    final String[] TEMPLATE_BODIES = new String[] {
        "This is a good template body {!Contact.Name}",
        "This is a bad template body {!Opportunity.Name} {!Contact.SNARF} ",
        "This is another good template body {!Contact.Name}";

    // Create request and add template bodies, whatId, and whoId.
    RenderEmailTemplateRequest req = new RenderEmailTemplateRequest();
    req.setTemplateBodies(TEMPLATE_BODIES);
    req.setWhatId(whatId);
    req.setWhoId(whoId);
    // An array of results is returned, one for each request.
    // We only have one request.
    RenderEmailTemplateResult[] results = connection.renderEmailTemplate(
        new RenderEmailTemplateRequest[] { req });
    if (results != null) {
        // Check results for our one and only request.
        // Check request was processed successfully, and if not, print the errors.
        if (!results[0].isSuccess()) {
            System.out.println(
                "The following errors were encountered while rendering email templates:");
            for (Error err : results[0].getErrors()) {
                System.out.println(err.getMessage());
            }
        } else {
            // Check results for each body template and print merged body
            RenderEmailTemplateBodyResult[] bodyResults = results[0].getBodyResults();
            for( Integer i=0;i<bodyResults.length;i++) {
                RenderEmailTemplateBodyResult result = bodyResults[i];
                if (result.isSuccess()) {
                    System.out.println("\nMerged body: \n" + result.getMergedBody());
                } else {
                    System.out.println("\nErrors were found for body[" + i + "]: ");
                    for (RenderEmailTemplateError err : result.getErrors()) {
                        System.out.println(err.getMessage() + " - Field name: "
                            + err.getFieldName());
                    }
                }
            }
        }
    }
}
```

Let's say you run this sample by specifying a valid contact ID for the first argument (*whoId*) and `null` for the second argument (*whatId*). The second template has one error set, for the incorrect merge field. The response looks like the following.

```
Merged body:
This is a good template body Howard Jones

Errors were found for body[1]:
Field Contact.SNARF does not exist. Check spelling. - Field name: Contact.SNARF

Merged body:
This is another good template body Howard Jones
```

RenderEmailTemplateRequest

Name	Type	Description
templateBodies	string[]	An array of text bodies that can contain merge fields, such as <code>{!Account.Phone}</code> or <code>{!Contact.Name}</code> .
whatId	reference	References a non-human object, such as an account, an opportunity, a campaign, a case, or a custom object. The <code>whatId</code> is polymorphic, which means that it's an ID that can refer to more than one type of object, such as a case or an opportunity.
whoId	reference	References a human object, such as a lead, contact, or user. The <code>whoId</code> is polymorphic, which means that it's an ID that can refer to more than one type of object.

Fault

The `renderEmailTemplate()` can return any of these API status codes.

[EMAIL_TEMPLATE_FORMULA_ERROR](#)

[EMAIL_TEMPLATE_MERGEFIELD_ACCESS_ERROR](#)

[EMAIL_TEMPLATE_MERGEFIELD_ERROR](#)

[EMAIL_TEMPLATE_MERGEFIELD_VALUE_ERROR](#)

[EMAIL_TEMPLATE_PROCESSING_ERROR](#)

RenderEmailTemplateResult

Contains status and error information for a request processed by the `renderEmailTemplate()` call, including individual results of rendered email templates.

Name	Type	Description
bodyResults	RenderEmailTemplateBodyResult[]	Contains status and error information for each template body that <code>renderEmailTemplate()</code> processed in a request, including merged body text of templates.

Name	Type	Description
errors	Error[]	Contains one or more errors that occurred when renderEmailTemplate() rendered a request.
success	boolean	Indicates whether a request was successfully processed (<code>true</code>) or not (<code>false</code>).

RenderEmailTemplateBodyResult

Contains status and error information for each template body that `renderEmailTemplate()` processed in a request, including merged body text of templates.

Name	Type	Description
errors	RenderEmailTemplateError []	Contains one or more errors that are associated with a template body that renderEmailTemplate() processed.
mergedBody	string	The text of the template body with the merge fields replaced with their corresponding values from Salesforce objects. The <code>whatId</code> and <code>whoId</code> fields on the request reference the Salesforce objects to use. The <code>mergedBody</code> field is populated only when the rendering of the template was successful (<code>success</code> is equal to <code>true</code>). If <code>success</code> is equal to <code>false</code> , <code>mergedBody</code> is null.
success	boolean	Indicates whether a template body was successfully rendered (<code>true</code>) or not (<code>false</code>).

RenderEmailTemplateError

An error that occurred when [renderEmailTemplate\(\)](#) processed a template body.

Name	Type	Description
fieldName	string[]	The merge field that affected the error condition.
message	string	Error message text.
offset	int	The offset in the template body text of the merge field that caused the error. The offset is calculated as the number of characters from the start of the body text. The offset is -1 if it can't be determined because of insufficient contextual information.
statusCode	StatusCode	A code that characterizes the error. The full list of status codes is available in the WSDL file for your organization (see Generating the WSDL File for Your Organization).

resetPassword()

Changes a user's password to a temporary, system-generated value.

Syntax

```
string password = connection.resetPassword(ID userID);
```

Usage

Use `resetPassword()` to request that the API change the password of a [User](#) or [SelfServiceUser](#), and return a system-generated password string of random letters and numbers. Use `setPassword()` instead if you want to set the password to a specific value.

Your client application must be logged in with sufficient access rights to change the password for the specified user. For more information, see [Factors that Affect Data Access](#).

For information on IDs, see [ID Field Type](#).

Sample Code—Java

This sample resets the password for the user specified by the `userId` parameter. It calls `resetPassword()` with this ID and gets the temporary password from the call result. It writes this temporary password to the console and returns it.

```
public String doResetPassword(String userId) {
    String result = "";
    try {
        ResetPasswordResult rpr = connection.resetPassword(userId);
        result = rpr.getPassword();
        System.out.println("The temporary password for user ID " + userId
            + " is " + result);
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
    return result;
}
```

Sample Code—C#

This sample resets the password for the user specified by the `userId` parameter. It calls `resetPassword()` with this ID and gets the temporary password from the call result. It writes this temporary password to the console and returns it.

```
public String doResetPassword(String userId)
{
    String result = "";
    try
    {
        ResetPasswordResult rpr = binding.resetPassword(userId);
        result = rpr.password;
        Console.WriteLine("The temporary password for user ID " + userId + " is " +
            result);
    }
}
```



```

catch (SoapException e)
{
    Console.WriteLine("An unexpected error has occurred: " +
        e.Message + "\n" + e.StackTrace);
}
return result;
}

```

Arguments

Name	Type	Description
userID	ID	ID of the User or SelfServiceUser whose password you want to reset. For information on IDs, see ID Field Type .

Response

Name	Type	Description
password	string	New password generated by the API. Once the user logs in with this password, they will be asked to provide a new password. This password is temporary, meaning that it cannot be reused once the user has set his or her new password.

Fault

[InvalidIdFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[Utility Calls](#)

sendEmail ()

Immediately sends an email message.

Syntax

For single email messages:

```
SendEmailResult = connection.sendEmail (SingleEmailMessage emails[]);
```

For mass email messages:

```
SendEmailResult = connection.sendEmail (MassEmailMessage emails[]);
```

Usage

Use this call with Lightning Platform AppExchange applications, custom applications, or other applications outside of Salesforce to send individual and mass email. The email can include all standard email attributes (such as subject line and blind carbon copy address), use Salesforce email templates, and be in plain text or HTML format. You can use Salesforce to track the status of HTML email, including the date the email was sent, first opened, last opened, and the total number of times it was opened. (See “Tracking HTML Email” in Salesforce Help for more information.)

The email address of the logged-in user is inserted in the `From Address` field of the email header. All return email and out-of-office replies go to the logged-in user. If bounce management is enabled and `SingleEmailMessage.targetObjectId` or `MassEmailMessage.targetObjectIds` is set, bounces are processed by Salesforce automatically, and the appropriate records are updated; otherwise, they go to the logged-in user. Bounce management works for contacts and leads only.

Note:

- Single email messages sent with this call count against the sending organization's daily single email limit. When this limit is reached, `sendEmail()` calls using `SingleEmailMessage` are rejected, and the user receives a `SINGLE_EMAIL_LIMIT_EXCEEDED` error code. However, single emails sent through the application are allowed.
- Mass email messages sent with this call count against the sending organization's daily mass email limit. When this limit is reached, `sendEmail()` calls using `MassEmailMessage` are rejected, and the user receives a `MASS_MAIL_LIMIT_EXCEEDED` error code.
- Starting in API version 35.0, you can enforce or ignore the **Email Opt Out** setting for contacts or leads with the `optOutPolicy` field of `SingleEmailMessage`. The `optOutPolicy` field applies to recipients in the To, CC, and BCC lists of the email. By default and in earlier versions, `SingleEmailMessage` ignores the **Email Opt Out** setting of recipients and the email is sent to all recipients. When using `MassEmailMessage`, the **Email Opt Out** setting of the recipients is always enforced—emails aren't sent to recipients that have opted out and are sent to all other recipients.

`SingleEmailMessage` has an optional field called `OrgWideEmailAddressId`. This is an object ID to an `OrgWideEmailAddress` object. If `OrgWideEmailAddressId` is set, the `OrgWideEmailAddress.DisplayName` field is used in the email header, instead of the logged-in user's `DisplayName`. The sending email address in the header is also set to the field defined in `OrgWideEmailAddress.Address`.



Note: If both the `DisplayName` in an `OrgWideEmailAddress` and `senderDisplayName` are defined, the user receives a `DUPLICATE_SENDER_DISPLAY_NAME` error.

Sample Code—Java

This sample creates an email message and sets its fields, including the To, CC and BCC recipients, subject, and body text. It also sets a recipient to the ID of the logged-in user using the `setTargetObjectId` method, which causes the email to be sent to the email address of the specified user. The sample creates an attachment and sends the email message with the attachment. Finally, it writes a status message or an error message, if any, to the console.

```
public void doSendEmail() {
    try {
        EmailFileAttachment efa = new EmailFileAttachment();
        byte[] fileBody = new byte[1000000];
        efa.setBody(fileBody);
        efa.setFileName("attachment");
        SingleEmailMessage message = new SingleEmailMessage();
        message.setBccAddresses(new String[] {
            "someone@salesforce.com"
        });
    };
```

```

message.setCcAddresses(new String[] {
    "person1@salesforce.com", "person2@salesforce.com", "003xx00000a1b2cAAC"
});
message.setBccSender(true);
message.setEmailPriority(EmailPriority.High);
message.setReplyTo("person1@salesforce.com");
message.setSaveAsActivity(false);
message.setSubject("This is how you use the " + "sendEmail method.");
// We can also just use an id for an implicit to address
getUserInfoResult guir = connection.getUserInfo();
message.setTargetObjectId(guir.getUserId());
message.setUseSignature(true);
message.setPlainTextBody("This is the humongous body "
    + "of the message.");
EmailFileAttachment[] efas = { efa };
message.setFileAttachments(efas);
message.setToAddresses(new String[] { "person3@salesforce.com" });
SingleEmailMessage[] messages = { message };
SendEmailResult[] results = connection.sendEmail(messages);
if (results[0].isSuccess()) {
    System.out.println("The email was sent successfully.");
} else {
    System.out.println("The email failed to send: "
        + results[0].getErrors()[0].getMessage());
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

This example shows how to send an email with the opt-out setting enforced. Recipients are specified by their IDs. The `SendEmailOptOutPolicy.FILTER` option causes the email to be sent only to recipients that haven't opted out from email.

```

SingleEmailMessage message = new SingleEmailMessage();
// Set recipients to two contact IDs.
// Replace IDs with valid record IDs in your org.
message.setToAddresses(new String[] { "003D000000QDexS", "003D000000QDfw5" });
message.setOptOutPolicy(SendEmailOptOutPolicy.FILTER);
message.setSubject("Opt Out Test Message");
message.setPlainTextBody("This is the message body.");
SingleEmailMessage[] messages = { message };
SendEmailResult[] results = connection.sendEmail(messages);
if (results[0].isSuccess()) {
    System.out.println("The email was sent successfully.");
} else {
    System.out.println("The email failed to send: "
        + results[0].getErrors()[0].getMessage());
}
}

```

Sample Code—C#

This sample creates an email message and sets its fields, including the To, CC and BCC recipients, subject, and body text. It also sets a recipient to the ID of the logged-in user using the `setTargetObjectId` method, which causes the email to be sent to the email

address of the specified user. The sample creates an attachment and sends the email message with the attachment. Finally, it writes a status message or an error message, if any, to the console.

```
public void doSendEmail()
{
    try
    {
        EmailFileAttachment efa = new EmailFileAttachment();
        byte[] fileBody = new byte[1000000];
        efa.body = fileBody;
        efa.fileName = "attachment";
        SingleEmailMessage message = new SingleEmailMessage();
        message.setBccAddresses(new String[] {
            "someone@salesforce.com"
        });
        message.setCcAddresses(new String[] {
            "person1@salesforce.com", "person2@salesforce.com", "003xx00000a1b2cAAC"
        });
        message.bccSender = true;
        message.emailPriority = EmailPriority.High;
        message.replyTo = "person1@salesforce.com";
        message.saveAsActivity = false;
        message.subject = "This is how you use the " + "sendEmail method.";
        // We can also just use an id for an implicit to address
        GetUserInfoResult guir = binding.getUserInfo();
        message.targetObjectId = guir.userId;
        message.useSignature = true;
        message.plainTextBody = "This is the humongous body "
            + "of the message.";
        EmailFileAttachment[] efas = { efa };
        message.fileAttachments = efas;
        message.toAddresses = new String[] { "person3@salesforce.com" };
        SingleEmailMessage[] messages = { message };
        SendEmailResult[] results = binding.sendEmail(messages);
        if (results[0].success)
        {
            Console.WriteLine("The email was sent successfully.");
        }
        else
        {
            Console.WriteLine("The email failed to send: "
                + results[0].errors[0].message);
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```


This example shows how to send an email with the opt-out setting enforced. Recipients are specified by their IDs. The `SendEmailOptOutPolicy.FILTER` option causes the email to be sent only to recipients that haven't opted out from email.

```
SingleEmailMessage message = new SingleEmailMessage();
// Set recipients to two contact IDs.
// Replace IDs with valid record IDs in your org.
message.toAddresses = new String[] { "003D000000QDexS", "003D000000QDfW5" };
message.optOutPolicy = SendEmailOptOutPolicy.FILTER;
message.subject = "Opt Out Test Message";
message.plainTextBody = "This is the message body.";
SingleEmailMessage[] messages = { message };
SendEmailResult[] results = binding.sendEmail(messages);
if (results[0].success)
{
    Console.WriteLine("The email was sent successfully.");
} else {
    Console.WriteLine("The email failed to send: "
        + results[0].errors[0].message);
}
```

BaseEmail

The following table contains the arguments used in both single and mass email.

 **Note:** If templates aren't being used, all email content must be in plain text, HTML, or both.

Name	Type	Description
<code>bccSender</code>	boolean	Indicates whether the email sender receives a copy of the email that is sent. For a mass mail, the sender is only copied on the first email sent.  Note: If the BCC compliance option is set at the organization level, the user can't add BCC addresses on standard messages. The following error code is returned: <code>BCC_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED</code> . Contact your Salesforce representative for information on BCC compliance.
<code>saveAsActivity</code>	boolean	Optional. The default value is <code>true</code> , meaning the email is saved as an activity. This argument only applies if the recipient list is based on <code>targetObjectId</code> or <code>targetObjectIds</code> . If HTML email tracking is enabled for the organization, you can track open rates.
<code>useSignature</code>	boolean	Indicates whether the email includes an email signature if the user has one configured. The default is <code>true</code> , meaning if the user has a signature it is included in the email unless you specify <code>false</code> .
<code>emailPriority</code>	picklist	Optional. The priority of the email. <ul style="list-style-type: none"> • Highest • High • Normal • Low


Name	Type	Description
		<ul style="list-style-type: none"> Lowest <p>The default is Normal.</p>
replyTo	string	Optional. The email address that receives the message when a recipient replies. This can't be set if you're using a Visualforce email template that specifies a replyTo value.
subject	string	Optional. The email subject line. If you're using an email template and attempt to override the subject line, an error message is returned.
templateId	ID	The ID of the template to be merged to create this email.
senderDisplayName	string	Optional. The name that appears on the From line of the email. This can't be set if the object associated with a OrgWideEmailAddressId for a SingleEmailMessage has defined its DisplayName field.

SingleEmailMessage

The following table contains the arguments single email uses in addition to the base email arguments.

Name	Type	Description
bccAddresses	string[]	<p>Optional. An array of blind carbon copy (BCC) addresses or object IDs of the contacts, leads, and users you're sending the email to. This argument is allowed only when a template isn't used. The maximum size for this field is 4,000 bytes. The maximum total of toAddresses, ccAddresses, and bccAddresses per email is 150. All recipients in these three fields count against the limit for email sent using Apex or the API.</p> <p>You can specify opt-out email options with the optOutPolicy field only for those recipients who were added by their IDs.</p> <p>Email addresses are verified to ensure that they have the correct format and haven't been marked as bounced.</p> <p>If the BCC COMPLIANCE option is set at the organization level, the user can't add BCC addresses on standard messages. The following error code is returned: BCC_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED.</p> <p>All emails must have a recipient value in at least one of the following fields:</p> <ul style="list-style-type: none"> toAddresses ccAddresses bccAddresses targetObjectId
ccAddresses	string[]	Optional. An array of carbon copy (CC) addresses or object IDs of the contacts, leads, and users you're sending the email to. This

Name	Type	Description
		<p>argument is allowed only when a template isn't used. The maximum size for this field is 4,000 bytes. The maximum total of <code>toAddresses</code>, <code>ccAddresses</code>, and <code>bccAddresses</code> per email is 150. All recipients in these three fields count against the limit for email sent using Apex or the API.</p> <p>You can specify opt-out email options with the <code>optOutPolicy</code> field only for those recipients who were added by their IDs.</p> <p>Email addresses are verified to ensure that they have the correct format and haven't been marked as bounced.</p> <p>All emails must have a recipient value in at least one of the following fields:</p> <ul style="list-style-type: none"> • <code>toAddresses</code> • <code>ccAddresses</code> • <code>bccAddresses</code> • <code>targetObjectId</code>
<code>charset</code>	string	Optional. The character set for the email. If this value is null, the user's default value is used. Unavailable if specifying <code>templateId</code> because the template specifies the character set.
<code>documentAttachments</code>	ID[]	Deprecated. Use entityAttachments instead. Optional. An array listing the ID of each Document you want to attach to the email.
<code>entityAttachments</code>	ID[]	Optional. Array of IDs of Document , ContentVersion , or Attachment items to attach to the email. This field is available in API version 35.0 and later.
<code>fileAttachments</code>	EmailFileAttachment[]	Optional. An array listing the file names of the binary and text files you want to attach to the email. You can attach multiple files as long as the total size of all attachments doesn't exceed 20 MB.
<code>htmlBody</code>	string	Optional. The HTML version of the email, specified by the sender. The value is encoded according to the specification associated with the organization.
<code>inReplyTo</code>	string	Optional. The In-Reply-To field of the outgoing email. Identifies the emails to which this one is a reply (parent emails). Contains the parent emails' Message-IDs. See RFC2822 - Internet Message Format .
<code>optOutPolicy</code>	SendEmailOptOutPolicy (enumeration of type string)	Optional. If you add contact, lead, or person account recipients by ID instead of email address, this field determines the behavior of the <code>sendEmail()</code> call. By default, the opt-out settings for recipients added by their email addresses aren't checked and those recipients always receive the email. Possible values of the <code>SendEmailOptOutPolicy</code> enumeration are:

Name	Type	Description
		<ul style="list-style-type: none"> SEND (default)—The email is sent to all recipients. The recipients' <code>Email Opt Out</code> setting is ignored. The setting <code>Enforce email privacy settings</code> is ignored. FILTER—No email is sent to recipients that have the <code>Email Opt Out</code> option set. Emails are sent to the other recipients. The setting <code>Enforce email privacy settings</code> is ignored. REJECT—If any of the recipients have the <code>Email Opt Out</code> option set, <code>sendEmail()</code> throws an error and no email is sent. The setting <code>Enforce email privacy settings</code> is respected, as are the selections in the data privacy record based on the <code>Individual</code> object. If any of the recipients have <code>Don't Market</code>, <code>Don't Process</code>, or <code>Forget this Individual</code> selected, <code>sendEmail()</code> throws an error and no email is sent. <p> Note: The <code>Send Non-Commercial Email</code> permission isn't respected.</p> <p>This field is available in API version 35.0 and later.</p>
<code>orgWideEmailAddressId</code>	ID	Optional. The object ID of the <code>OrgWideEmailAddress</code> associated with the outgoing email. <code>OrgWideEmailAddress.DisplayName</code> can't be set if the <code>senderDisplayName</code> field is already set.
<code>plainTextBody</code>	string	Optional. The text version of the email, specified by the sender.
<code>references</code>	string	Optional. The <code>References</code> field of the outgoing email. Identifies an email thread. Contains the parent emails' <code>Message-ID</code> and <code>References</code> fields and possibly <code>In-Reply-To</code> fields. See RFC2822 - Internet Message Format .
<code>targetObjectId</code>	ID	Optional. The object ID of the contact, lead, or user the email will be sent to. The object ID you enter sets the context and ensures that merge fields in the template contain the correct data All emails must have a recipient value in at least one of the following fields: <ul style="list-style-type: none"> <code>toAddresses</code> <code>ccAddresses</code> <code>bccAddresses</code> <code>targetObjectId</code>
<code>toAddresses</code>	string[]	Optional. An array of email addresses or object IDs of the contacts, leads, or users you're sending the email to. This argument is allowed only when a template isn't used. The maximum size for this field is 4,000 bytes. The maximum total of <code>toAddresses</code> , <code>ccAddresses</code> , and <code>bccAddresses</code> per email is 150. All recipients in these three fields count against the limit for email sent using Apex or the API.

Name	Type	Description
		<p>You can specify opt-out email options with the <code>optOutPolicy</code> field only for those recipients who were added by their IDs.</p> <p>Email addresses are verified to ensure that they have the correct format and haven't been marked as bounced.</p> <p>All emails must have a recipient value in at least one of the following fields:</p> <ul style="list-style-type: none"> • <code>toAddresses</code> • <code>ccAddresses</code> • <code>bccAddresses</code> • <code>targetObjectId</code>
<code>treatBodiesAsTemplate</code>	boolean	<p>Optional. If set to <code>true</code>, the subject, plain text, and HTML text bodies of the email are treated as template data. The merge fields are resolved using the <code>renderEmailTemplate()</code> call. Default is <code>false</code>.</p> <p>This field is available in API version 35.0 and later.</p>
<code>treatTargetObjectAsRecipient</code>	boolean	<p>Optional. If set to <code>true</code>, the <code>targetObjectId</code> (a contact, lead, or user) is the recipient of the email. If set to <code>false</code>, the <code>targetObjectId</code> is supplied as the <code>whoId</code> field for template rendering but isn't a recipient of the email. The default is <code>true</code>.</p> <p>This field is available in API version 35.0 and later. In prior versions, the <code>targetObjectId</code> is always a recipient of the email.</p>
<code>whatId</code>	ID	<p>Optional. If you specify a contact for the <code>targetObjectId</code> field, you can specify a <code>whatId</code> as well. This field helps to further ensure that merge fields in the template contain the correct data. The value must be one of the following types:</p> <ul style="list-style-type: none"> • <code>Account</code> • <code>Asset</code> • <code>Campaign</code> • <code>Case</code> • <code>Contract</code> • <code>Opportunity</code> • <code>Order</code> • <code>Product</code> • <code>Solution</code> • <code>Custom</code>

MassEmailMessage

The following table contains the arguments mass email uses in addition to the base email arguments.

Name	Type	Description
<code>description</code>	string	A value used internally to identify the object in the mass email queue.
<code>targetObjectIds</code>	ID[]	An array of object IDs of the contacts, leads, or users the email will be sent to. The object IDs you enter set the context and ensure that merge fields in the template contain the correct data. The objects must be of the same type (either all contacts, all leads, or all users). You can list up to 250 IDs per email. If you specify a value for the <code>targetObjectIds</code> field, optionally specify a <code>whatId</code> as well to set the email context to a user, contact, or lead. This ensures that merge fields in the template contain the correct data.
<code>whatIds</code>	ID[]	Optional. If you specify an array of contacts for the <code>targetObjectIds</code> field, you can specify an array of <code>whatIds</code> as well. This helps to further ensure that merge fields in the template contain the correct data. The values must be one of the following types: <ul style="list-style-type: none"> • Contract • Case • Opportunity • Product If you specify <code>whatIds</code> , specify one for each <code>targetObjectId</code> ; otherwise, you receive an <code>INVALID_ID_FIELD</code> error.

EmailFileAttachment

The following table contains properties that the `EmailFileAttachment` uses in the `SingleEmailMessage` object to specify attachments passed in as part of the request, as opposed to a [Document](#) passed in using the `documentAttachments` argument.

Property	Type	Description
<code>body</code>	base64	The attachment itself.
<code>contentType</code>	string	Optional. The attachment's Content-Type.
<code>fileName</code>	string	The name of the file to attach.
<code>inline</code>	boolean	Optional. Specifies a Content-Disposition of inline (<code>true</code>) or attachment (<code>false</code>). In most cases, inline content is displayed to the user when the message is opened. Attachment content requires user action to be displayed.

Response

[SendEmailResult](#)

Fault

The following API status codes can be returned. Also, `sendEmail()` can return other errors when rendering email templates. See `renderEmailTemplate()` Faults.

`BCC_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED`
`BCC_SELF_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED`
`DUPLICATE_SENDER_DISPLAY_NAME`
`EMAIL_ADDRESS_BOUNCED`
`EMAIL_NOT_PROCESSED_DUE_TO_PRIOR_ERROR`
`EMAIL_OPTED_OUT`
`ERROR_IN_MAILER`
`INSUFFICIENT_ACCESS_ON_CROSS_REFERENCE_ENTITY`
`INVALID_CONTENT_TYPE`
`INVALID_EMAIL_ADDRESS`
`INVALID_ID_FIELD`
`INVALID_MESSAGE_ID_REFERENCE`
`INVALID_SAVE_AS_ACTIVITY_FLAG`
`LIMIT_EXCEEDED`
`MALFORMED_ID`
`MASS_MAIL_LIMIT_EXCEEDED`
`NO_MASS_MAIL_PERMISSION`
`REQUIRED_FIELD_MISSING`
`SINGLE_EMAIL_LIMIT_EXCEEDED`
`TEMPLATE_NOT_ACTIVE`
`UNVERIFIED_SENDER_ADDRESS`

SendEmailResult

The `sendEmail()` call returns a list of `SendEmailResult` objects. Each `SendEmailResult` object has the following properties:


Name	Type	Description
<code>success</code>	boolean	<p>If sending single email: Indicates whether the email was successfully accepted for delivery by the message transfer agent (<code>true</code>) or not (<code>false</code>). Even if <code>success = true</code>, it does not mean the intended recipients received the email, as it could have bounced or been blocked by a spam blocker. Also, even if the email is successfully accepted for delivery by the message transfer agent, there can still be errors in the error array related to individual addresses within the email.</p> <p>If sending mass email: Indicates whether the email was successfully added to the queue for processing (<code>true</code>) or not (<code>false</code>). Even if the email was added to the</p>

Name	Type	Description
		queue, there can still be processing errors that prevent delivery to the intended recipients.
<code>SendEmailError</code>	<code>Error[]</code>	If an error occurred during the <code>sendEmail()</code> call, a list of <code>SendEmailError</code> objects is returned. For single email, errors indicate that Salesforce wasn't able to deliver the email. For mass email, errors indicate that the email wasn't added to the queue for processing.

SendEmailError

SendEmailError can have the following attributes:

Name	Type	Description
Fields	<code>Field[]</code>	Reserved for future use. Array of one or more field names. Identifies which fields in the object, if any, affected the error condition.
Message	<code>string</code>	Error message text.
StatusCode	<code>statusCode</code>	A code that characterizes the error. The full list of status codes is available in the WSDL file for your organization .
TargetObjectId	<code>ID</code>	The object ID of the target for which the error occurred.

 **Note:** If an error occurs that prevents `sendEmail()` from sending the email to one or more targets, each `TargetObjectId` for those targets has an associated error in `SendEmailResult`. A `TargetObjectId` that does not have an associated error in `SendEmailResult` indicates the email was sent to the target. If `SendEmailResult` has an error that does not have an associated `TargetObjectId`, no email was sent.

The following is an example of how to parse through a resulting set for errors:

```
Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
email.setToAddresses(new String[] { 'admin@acme.com' });
email.setSubject('my subject');
email.setPlainTextBody('plain text body');
List<Messaging.SendEmailResult> results =
    Messaging.sendEmail(new Messaging.Email[] { email });
if (!results.get(0).isSuccess()) {
    System.StatusCode statusCode = results.get(0).getErrors()[0].getStatusCode();
    String errorMessage = results.get(0).getErrors()[0].getMessage();
}
```

sendEmailMessage()

Immediately sends up to 10 draft email messages.

Syntax

For Enterprise SOAP:

```
SendEmailResult[] = connection.sendEmailMessage( String[] draftEmailIds);
```

For Partner SOAP:

```
SendEmailResult[] = connection.sendEmailMessage( ID[] draftEmailIds);
```

Usage

Use this call with Lightning Platform AppExchange applications, custom applications, or other applications outside of Salesforce to send up to 10 draft email messages. The messages can include all standard email attributes (such as subject line and blind carbon copy address), use Salesforce email templates, and be in plain text or HTML format. You can use Salesforce to track the status of HTML email, including the date the email was sent, first opened, last opened, and the total number of times it was opened. (See “Tracking HTML Email” in the Salesforce online help for more information.)

The email address of the logged-in user is inserted in the `From Address` field of the email header. All return email and out-of-office replies go to the logged-in user. If bounce management is enabled and `SingleEmailMessage.targetObjectId` or `MassEmailMessage.targetObjectIds` is set, bounces are processed by Salesforce automatically, and the appropriate records are updated; otherwise, they go to the logged-in user. Bounce management works for contacts and leads only.

Note:

- Email messages sent with this call count against the sending organization's daily single email limit. When this limit is reached, `sendEmailMessage()` calls using `SingleEmailMessage` are rejected, and the user receives a `SINGLE_EMAIL_LIMIT_EXCEEDED` error code. However, single emails sent through the application are allowed.
- Mass email messages sent with this call count against the sending organization's daily mass email limit. When this limit is reached, `sendEmail()` calls using `MassEmailMessage` are rejected, and the user receives a `MASS_MAIL_LIMIT_EXCEEDED` error code.
- The `AllOrNone` header is not honored by this call. `sendEmailMessage()` returns partial success even if the `AllOrNone` header is set to `true`.

Sample Code—Java

This sample creates a case and a draft email message, and sets the message fields, including the `From`, `To`, `CC`, and `BCC` recipients, subject, and body text. It also creates an attachment and sends the email message with the attachment. Finally, it writes a status message or an error message, if any, to the console.

```
public void doSendEmail() {
    try {
        //Create a case
        Case theCase = new Case();
        theCase.setSubject("Sample Case");
        SaveResult[] saveResult = connection.create(new SObject[] { theCase });
        String caseId = saveResult[0].getId();

        //Create a draft EmailMessage
        EmailMessage message = new EmailMessage();
        message.setParentId(theCase.getId());
        message.setBccAddress("bcc@email.com");
    }
}
```

```

message.setCcAddress("cc1@salesforce.com; cc2@email.com");
message.setSubject("This is how you use the sendEmailMessage method.");
message.setFromAddress("from@email.com");
message.setFromName("Sample Code");
message.setTextBody("This is the text body of the message.");
message.setStatus("5"); // "5" means Draft
message.setToAddress("to@email.com");
saveResult = connection.create(new SObject[] { message });
String emailMessageId = saveResult[0].getId();

//Create an attachment for the draft EmailMessage
Attachment att = new Attachment();
byte[] fileBody = new byte[1000000];
att.setBody(fileBody);
att.setName("attachment");
att.setParentId(emailMessageId);
connection.create(new SObject[] { att });

//Send the draft EmailMessage
SendEmailResult[] results = connection.sendEmailMessage(messages);
if (results[0].isSuccess()) {
    System.out.println("The email was sent successfully.");
} else {
    System.out.println("The email failed to send: " +
        results[0].getErrors()[0].getMessage());
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}

```

Arguments

None.

Response

SendEmailResult[]

Fault

[BCC_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED](#)
[BCC_SELF_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED](#)
[EMAIL_NOT_PROCESSED_DUE_TO_PRIOR_ERROR](#)
[ERROR_IN_MAILER](#)
[INSUFFICIENT_ACCESS_ON_CROSS_REFERENCE_ENTITY](#)
[INVALID_CONTENT_TYPE](#)
[INVALID_EMAIL_ADDRESS](#)
[INVALID_ID_FIELD](#)

INVALID_MESSAGE_ID_REFERENCE
 LIMIT_EXCEEDED
 MALFORMED_ID
 REQUIRED_FIELD_MISSING
 SINGLE_EMAIL_LIMIT_EXCEEDED
 TEMPLATE_NOT_ACTIVE
 UNVERIFIED_SENDER_ADDRESS

setPassword()

Sets the specified user's password to the specified value.

Syntax

```
SetPasswordResult setPasswordResult = connection.setPassword(ID userID, string password);
```

Usage

Use `setPassword()` to change the password of a [User](#) or [SelfServiceUser](#) to a value that you specify. For example, a client application might prompt a user to specify a different password, and then invokes `setPassword()` for an admin to change the user's password. Use `resetPassword()` instead if you want to reset the password with a random value generated by the API.

This call can be used to allow users to change their own passwords, as long as their org's Password Policies setting **Allow use of setPassword() API for self-resets** is enabled. Otherwise, use `changeOwnPassword()`, which is more secure because it verifies the user's current password before allowing the change.

Your client application must be logged in with sufficient access rights to change the password for the specified user. For more information, see [Factors that Affect Data Access](#).

For information on IDs, see [ID Field Type](#).

This call can use the session ID returned in [LoginResult](#) if the password has expired. For more information, see [passwordExpired](#).

Sample Code—Java

This sample accepts user ID and password parameters, which it uses in the `setPassword()` call to set the password of the specified user.

```
public void doSetPassword(String userID, String newPasswd) {
    try {
        SetPasswordResult result = connection.setPassword(userID, newPasswd);
        System.out.println("The password for user ID " + userID + " changed to "
            + newPasswd);
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

Sample Code—C#

This sample accepts user ID and password parameters, which it uses in the `setPassword()` call to set the password of the specified user.

```
public void doSetPassword(String userId, String newPasswd)
{
    try
    {
        SetPasswordResult result = binding.setPassword(userId, newPasswd);
        Console.WriteLine("The password for user ID " + userId + " changed to "
            + newPasswd);
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

Arguments

Name	Type	Description
userID	ID	ID of the User or SelfServiceUser whose password you want to reset. For information on IDs, see ID Field Type .
password	string	New password to use for the specified user.

Response

SetPasswordResult (empty)

Fault

[InvalidIdFault](#)

[UnexpectedErrorFault](#)

SEE ALSO:

[resetPassword\(\)](#)

[Utility Calls](#)

[changeOwnPassword\(\)](#)

CHAPTER 12 SOAP Headers

The API provides SOAP headers to client applications.

Header	Description
AllOrNoneHeader	Specifies whether a call rolls back all changes unless all records are processed successfully. This header is available in API version 20.0 and later.
AllowFieldTruncationHeader	Specifies the truncation behavior for some field types in API version 15.0 and later.
AssignmentRuleHeader	Specifies the assignment rule to use when creating or updating an Account , Case , or Lead .
CallOptions	Specifies the call options for an API request.
DebuggingHeader	Returns the debug log in the output header, <code>DebuggingInfo</code> , and specifies the level of detail in the debug log.
DisableFeedTrackingHeader	Specifies whether the changes made in the current call are tracked in feeds.
DuplicateRuleHeader	Determines options for using duplicate rules to detect duplicate records. Duplicate rules are part of the Duplicate Management feature.
EmailHeader	Sends an email notification when a request is processed. Provides equivalent functionality for the Salesforce user interface.
LimitInfoHeader	A response header returned from calls to SOAP API. This header returns limit information for the organization. Use this header to monitor your API limits as you make calls against the organization.
LocaleOptions	Specifies the language of the labels returned. The value must be a valid user locale (language and country), such as <code>de_DE</code> or <code>en_GB</code> . For more information on locales, see the Language field on the <code>CategoryNodeLocalization</code> object.
LoginScopeHeader	Specifies the organization ID so that you can authenticate Self-Service users for your organization using the <code>login()</code> call.
MruHeader	Indicates whether to update the list of most recently used items (<code>true</code>) or not (<code>false</code>).
OwnerChangeOptions	Specifies ownership of attachments and notes.
PackageVersionHeader	Specifies the package version for each installed managed package in API version 16.0 and later.
QueryOptions	Specifies the batch size for query results.
SessionHeader	Specifies the session ID returned from the login server after a successful <code>login()</code> .

Header	Description
UserTerritoryDeleteHeader	Specifies a user to whom open opportunities are assigned when the current owner is removed from a territory.

AllOrNoneHeader

Allows a call to roll back all changes unless all records are processed successfully.

Without the AllOrNoneHeader header, records without errors are committed, while records with errors are marked as failed in the call results. This header is available in API version 20.0 and later.

Even if the header is enabled, it's still necessary to inspect the `success` field in the call result for each record to identify records with errors. Each `success` field contains `true` or `false` indicating whether the call was processed successfully.

If there is an error associated with at least one record, the `errors` field in the call result for the record gives more information on the error. If other records in the same call have no errors, their `errors` fields indicate that they were rolled back due to other errors.

API Calls

`create()`, `delete()`, `undelete()`, `update()`, `upsert()`

Fields

Element Name	Type	Description
<code>allOrNone</code>	boolean	<p>If <code>true</code>, any failed records in a call cause all changes for the call to be rolled back. Record changes aren't committed unless all records are processed successfully.</p> <p>The default is <code>false</code>. Some records can be processed successfully while others are marked as failed in the call results.</p>

Sample Code—Java

This sample shows how to use the `AllOrNoneHeader`. It attempts to create two contacts. The second contact doesn't have all required fields set and causes a failure on creation. Next, the sample sets the `allOrNone` field to `true`, and then attempts to create the contacts. Creating one of the contacts results in an error, so the entire transaction is rolled back and no contacts are created.

```
public void allOrNoneHeaderSample() {
    try {
        // Create the first contact.
        SObject[] sObjects = new SObject[2];
        Contact contact1 = new Contact();
        contact1.setFirstName("Robin");
        contact1.setLastName("Van Persie");

        // Create the second contact. This contact doesn't
        // have a value for the required
```

```

// LastName field so the create will fail.
Contact contact2 = new Contact();
contact2.setFirstName("Ashley");
sObjects[0] = contact1;
sObjects[1] = contact2;

// Set the SOAP header to roll back the create unless
// all contacts are successfully created.
connection.setAllOrNoneHeader(true);
// Attempt to create the two contacts.
SaveResult[] sr = connection.create(sObjects);
for (int i = 0; i < sr.length; i++) {
    if (sr[i].isSuccess()) {
        System.out.println("Successfully created contact with id: " +
            sr[i].getId() + ".");
    }
    else {
        // Note the error messages as the operation was rolled back
        // due to the all or none header.
        System.out.println("Error creating contact: " +
            sr[i].getErrors()[0].getMessage());
        System.out.println("Error status code: " +
            sr[i].getErrors()[0].getStatusCode());
    }
}
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

AllowFieldTruncationHeader

Specifies that for some fields, when a string is too large, the operation fails. Without the header, strings for these fields are truncated.

The `AllowFieldTruncationHeader` header affects the following datatypes:

- anyType, if it represents one of the other datatypes in this list
- email
- encryptedstring
- multipicklist
- phone
- picklist
- string
- textarea

In API versions previous to 15.0, if a value for one of the listed fields is too large, the value is truncated.

For API version 15.0 and later, if a value is too large, the operation fails and the fault code `STRING_TOO_LONG` is returned. `AllowFieldTruncationHeader` allows you to specify that the previous behavior, truncation, be used instead of the new behavior in API versions 15.0 and later.

This header has no effect in versions 14.0 and earlier.

API Calls

[convertLead\(\)](#), [create\(\)](#), [merge\(\)](#), [process\(\)](#), [undelete\(\)](#), [update\(\)](#), and [upsert\(\)](#)

Apex: [executeanonymous \(\)](#)

Fields

Element Name	Type	Description
<code>allowFieldTruncation</code>	boolean	<p>If <code>true</code>, truncate field values that are too long, which is the behavior in API versions 14.0 and earlier.</p> <p>Default is <code>false</code>: no change in behavior. If a <code>string</code> or <code>textarea</code> value is too large, the operation fails and the fault code <code>STRING_TOO_LONG</code> is returned.</p> <p>The following list shows the field types affected by truncation and this header:</p> <ul style="list-style-type: none"> • <code>anyType</code>, if it represents one of the other datatypes in this list • <code>email</code> • <code>encryptedstring</code> • <code>multipicklist</code> • <code>phone</code> • <code>picklist</code> • <code>string</code> • <code>textarea</code>

Sample Code—Java

To create an account with a name that is too long for the `Name` field, use the `AllowFieldTruncation` header.

This sample:

1. Creates an `Account` object with a name that exceeds the field limit of 255 characters.
2. Sends the create call, which fails because of the name field length.
3. Sets the `AllowFieldTruncationHeader` to `true` and retries the account creation, which succeeds.

```
public void allowFieldTruncationSample() {
    try {
        Account account = new Account();
        // Construct a string that is 256 characters long.
        // Account.Name's limit is 255 characters.
        String accName = "";
        for (int i = 0; i < 256; i++) {
            accName += "a";
        }
        account.setName(accName);
        // Construct an array of SObjects to hold the accounts.
```

```

SObject[] sObjects = new SObject[1];
sObjects[0] = account;
// Attempt to create the account. It will fail in API version 15.0
// and above because the account name is too long.
SaveResult[] results = connection.create(sObjects);
System.out.println("The call failed because: "
    + results[0].getErrors()[0].getMessage());
// Now set the SOAP header to allow field truncation.
connection.setAllowFieldTruncationHeader(true);
// Attempt to create the account now.
results = connection.create(sObjects);
System.out.println("The call: " + results[0].isSuccess());
} catch (ConnectionException ce) {
    ce.printStackTrace();
}
}
}

```

AssignmentRuleHeader

The `AssignmentRuleHeader` must be specified in the `create()` or `update()` call of a `Case` or `Lead` for the specified assignment rule to be applied, and it must be specified in the `update()` call of an `Account` for the territory assignment rules to be applied.

API Calls

`create()`, `merge()`, `update()`, `upsert()`

Fields

Element Name	Type	Description
<code>assignmentRuleId</code>	ID	The ID of a specific assignment rule to run for the Case or Lead. The assignment rule can be active or inactive. The ID can be retrieved by querying the <code>AssignmentRule</code> object. If specified, do not specify <code>useDefaultRule</code> . This element is ignored for accounts, because all territory assignment rules are applied. If the value is not in correct ID format (15-character or 18-character Salesforce ID), the call fails and a <code>MALFORMED_ID</code> exception is returned.
<code>useDefaultRule</code>	boolean	If <code>true</code> for a Case or Lead, uses the default (active) assignment rule for a Case or Lead. If specified, do not specify an <code>assignmentRuleId</code> . If <code>true</code> for an Account, all territory assignment rules are applied, and if <code>false</code> , no territory assignment rules are applied.

Sample Code

For a code example, see [Lead](#).

SEE ALSO:

[AssignmentRule](#)

CallOptions

Specifies the options needed to work with a specific client. This header is only available for use with the [Partner WSDL](#).

API Calls

The `defaultNamespace` element supports the following calls: [create\(\)](#), [merge\(\)](#), [queryAll\(\)](#), [query\(\)](#), [queryMore\(\)](#), [retrieve\(\)](#), [search\(\)](#), [update\(\)](#), and [upsert\(\)](#).

The `client` element supports all of the above calls, plus the following: [convertLead\(\)](#), [login\(\)](#), [delete\(\)](#), [describeGlobal\(\)](#), [describeLayout\(\)](#), [describeTabs\(\)](#), [describeSObject\(\)](#), [describeSObjects\(\)](#), [getDeleted\(\)](#), [getUpdated\(\)](#), [process\(\)](#), [undelete\(\)](#), [getServerTimestamp\(\)](#), [getUserInfo\(\)](#), [setPassword\(\)](#), and [resetPassword\(\)](#).

Fields

Element Name	Type	Description
<code>client</code>	string	A string that identifies a client.
<code>defaultNamespace</code>	string	<p>A string that identifies a developer namespace prefix. Use this field to resolve field names in managed packages without having to fully specify the <code>fieldName</code> everywhere.</p> <p>For example, if the developer namespace prefix is <code>battle</code>, and you have a custom field in your package called <code>botId</code>, you can set this header, and then queries such as the following will succeed:</p> <pre>query("SELECT id, botId__c from Account");</pre> <p>In this case the actual field queried is the <code>battle__botId__c</code> field.</p> <p>Using this field allows you to write client code without having to specify the namespace prefix. Without this field specified, the full name of the field would have to be used for the query to succeed. In the example above, you would have to specify <code>battle__botId__c</code>.</p> <p>Note that if this field is set, and the query specifies the namespace as well, the response will not include the prefix. For example, if you set this header to <code>battle</code>, and issue a query like <code>query("SELECT id, battle__botId__c from Account");</code>, the response would use a <code>botId__c</code> element, not a <code>battle__botId__c</code> element.</p> <p>Describe calls ignore this header, so there will be no ambiguity between fields with namespace prefixes and customer fields of the same name without the prefix.</p>

Sample Code—C#

This sample shows how to use the `CallOptions` header. It sets a client ID and a developer namespace prefix, which is used to resolve field names in managed packages. Next, the sample logs the specified user in.

```
public void CallOptionsSample()
{
    // Web Reference to the imported Partner WSDL.
    APISamples.partner.SforceService partnerBinding;

    string username = "USERNAME";
    string password = "PASSWORD";

    // The real Client ID will be an API Token provided by Salesforce
    // to partner applications following a security review.
    // For more details, see the Security Review FAQ in the online help.
    string clientId = "SampleCaseSensitiveToken/100";

    partnerBinding = new SforceService();
    partnerBinding.CallOptionsValue = new CallOptions();
    partnerBinding.CallOptionsValue.client = clientId;

    // Optionally, if a developer namespace prefix has been registered for
    // your Developer Edition organization, it may also be specified.
    string prefix = "battle";
    partnerBinding.CallOptionsValue.defaultNamespace = prefix;

    try
    {
        APISamples.partner.LoginResult lr =
            partnerBinding.login(username, password);
    }
    catch (SoapException e)
    {
        Console.WriteLine(e.Code);
        Console.WriteLine(e.Message);
    }
}
```

DisableFeedTrackingHeader

Specifies that changes made in the current call are tracked in feeds.

Use this header if you want to process many records without tracking the changes in various feeds related to the records. This header is available if the Chatter feature is enabled for your organization.

API Calls

`convertLead()`, `create()`, `delete()`, `merge()`, `process()`, `undelete()`, `update()`, `upsert()`

Fields

Element Name	Type	Description
disableFeedTracking	boolean	If <code>true</code> , the changes made in the current call are not tracked in feeds. The default is <code>false</code> .

Sample Code—Java

This sample shows how to use the `DisableFeedTrackingHeader`. It sets this header to `true` to disable feed tracking and then creates many account records in bulk.

```
public void disableFeedTrackingHeaderSample() {
    try {
        // Insert a large number of accounts.
        SObject[] sObjects = new SObject[500];
        for (int i = 0; i < 500; i++) {
            Account a = new Account();
            a.setName("my-account-" + i);
            sObjects[i] = a;
        }
        // Set the SOAP header to disable feed tracking to avoid generating a
        // large number of feed items because of this bulk operation.
        connection.setDisableFeedTrackingHeader(true);
        // Perform the bulk create. This won't result in 500 feed items, which
        // would otherwise be generated without the DisableFeedTrackingHeader.
        SaveResult[] sr = connection.create(sObjects);
        for (int i = 0; i < sr.length; i++) {
            if (sr[i].isSuccess()) {
                System.out.println("Successfully created account with id: " +
                    sr[i].getId() + ".");
            } else {
                System.out.println("Error creating account: " +
                    sr[i].getErrors()[0].getMessage());
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

SEE ALSO:

[NewsFeed](#)

[EntitySubscription](#)

DebuggingHeader

Return the debug log in the output header, `DebuggingInfo`, and specify the level of detail in the debug log.

 **Note:** Calls that include DebuggingHeader are limited to 1,000 in a 24-hour period. You can continue to make these calls even after reaching the total request limit for an org.

API Calls

`compileAndTest()`, `executeAnonymous()`, `runTests()`

Fields

Element Name	Type	Description
<code>categories</code>	<code>LogInfo[]</code>	Specifies the type and amount of information to be returned in the debug log.
<code>debugLevel</code>	<code>DebugLevel</code> (enumeration of type string)	<p>Deprecated. This field is provided only for backward compatibility. If you provide values for both <code>debugLevel</code> and <code>categories</code>, the <code>categories</code> value is used.</p> <p>The <code>debugLevel</code> field 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:</p> <ul style="list-style-type: none"> • None • Debugonly • Db • Profiling • Callout • Detail

LogInfo

Specifies the type and amount of information to be returned in the debug log. The `categories` field takes a list of these objects. LogInfo is a mapping of `category` to `level`.

Fields

Element Name	Type	Description
<code>category</code>	<code>LogCategory</code>	<p>Specify the type of information returned in the debug log. Valid values are:</p> <ul style="list-style-type: none"> • Db • Workflow • Validation • Callout • Apex_code • Apex_profiling • Visualforce

Element Name	Type	Description
		<ul style="list-style-type: none"> • System • All
level	LogCategoryLevel	<p>Specifies the level of detail returned in the debug log.</p> <p>Valid log levels are (listed from lowest to highest):</p> <ul style="list-style-type: none"> • NONE • ERROR • WARN • INFO • DEBUG • FINE • FINER • FINEST

DuplicateRuleHeader

Determines options for using duplicate rules to detect duplicate records. Duplicate rules are part of the Duplicate Management feature.

API Calls

[create\(\)](#), [update\(\)](#), [upsert\(\)](#)

Fields

Element Name	Type	Description
allowSave	boolean	For a duplicate rule, when the Alert option is enabled, bypass alerts and save duplicate records by setting this property to <code>true</code> . Prevent duplicate records from being saved by setting this property to <code>false</code> .
includeRecordDetails	boolean	Get fields and values for records detected as duplicates by setting this property to <code>true</code> . Get only record IDs for records detected as duplicates by setting this property to <code>false</code> .
runAsCurrentUser	boolean	Make sure that sharing rules for the current user are enforced when duplicate rules run by setting this property to <code>true</code> . Use the sharing rules specified in the class for the request by setting this property to <code>false</code> . If no sharing rules are specified, Apex code runs in system context and sharing rules for the current user are not enforced.

Java Sample

This sample shows how to use the `DuplicateRuleHeader` to set options for using duplicate rules. To see the entire sample application, see [DuplicateResult](#).

```
_DuplicateRuleHeader header = new _DuplicateRuleHeader();
    header.setAllowSave(false);
    header.setIncludeRecordDetails(true);
    header.setRunAsCurrentUser(true);

    binding.setHeader(new SforceServiceLocator().getServiceName().getNamespaceURI(),
        "DuplicateRuleHeader", header);
```

SEE ALSO:

[DuplicateResult](#)

[DuplicateRule](#)

EmailHeader

The Salesforce user interface allows you to specify whether to send an email when these events occur:

- Create a [Case](#)
- Create a [CaseComment](#)
- Convert Case email to a [Contact](#)
- Send a New [User](#) email notification
- Make a `resetPassword()` call

In API versions 8.0 and later, you can also send an API request that sends email.

A group event is an [Event](#) for which `IsGroupEvent` is true. The [EventRelation](#) object tracks the users, leads, or contacts that are invited to a group event. Note the following behaviors for group event email sent through the API:

- Sending a group event invitation to a [User](#) respects the `triggerUserEmail` option
- Sending a group event invitation to a [Lead](#) or [Contact](#) respects the `triggerOtherEmail` option
- Email sent when updating or deleting a group event also respect `triggerUserEmail` and `triggerOtherEmail`, as appropriate

API Calls

[create\(\)](#), [delete\(\)](#), [resetPassword\(\)](#), [update\(\)](#), [upsert\(\)](#)

Fields

Element Name	Type	Description
<code>triggerAutoResponseEmail</code>	boolean	Indicates whether to trigger auto-response rules (<code>true</code>) or not (<code>false</code>), for leads and cases. In the Salesforce user interface, this email can be automatically triggered by a number of events, for example creating a case or resetting a

Element Name	Type	Description
		user password. If this value is set to <code>true</code> , when a Case is created, if there is an email address for the contact specified in ContactId , the email is sent to that address. If not, the email is sent to the address specified in SuppliedEmail .
<code>triggerOtherEmail</code>	boolean	Indicates whether to trigger email outside the organization (<code>true</code>) or not (<code>false</code>). In the Salesforce user interface, this email can be automatically triggered by creating, editing, or deleting a contact for a case.
<code>triggerUserEmail</code>	boolean	Indicates whether to trigger email that is sent to users in the organization (<code>true</code>) or not (<code>false</code>). In the Salesforce user interface, this email can be automatically triggered by a number of events; resetting a password, creating a new user, or adding comments to a case.

Sample Code—Java

This sample shows how to use the `EmailHeader`. It sets the `triggerAutoResponseEmail` email header field to `true`, which triggers an email to be sent when a case is created. Next, it creates a case. This sample assumes an auto-response rule has been set for cases, and an email address is specified in the contact referenced by [ContactId](#).

```
public void createCaseWithAutoResponse(String contactId) {
    try {
        connection.setEmailHeader(true, false, false);
        Case c = new Case();
        c.setSubject("Sample Subject");
        c.setContactId(contactId);
        SaveResult[] sr = connection.create(new SObject[] { c });
        // Parse sr array to see if case was created successfully.
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

LimitInfoHeader

A response header returned from calls to SOAP API. This header returns limit information for the organization. Use this header to monitor your API limits as you make calls against the organization.

API Calls

All calls, except for `login()`.

Fields

Element Name	Type	Description
current	string	The number of calls for the specified limit type that have already been used in the organization.
limit	string	The organization's limit for the specified limit type.
type	string	The type of limit information specified in the header. <ul style="list-style-type: none"> API REQUESTS— the daily API usage for the organization against which the call was made.

Sample Code

This example shows a response to a SOAP request for a Merchandise record. The `LimitInfoHeader` contains the API usage information for the organization.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns="urn:partner.soap.sforce.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sf="urn:subject.partner.soap.sforce.com">
  <soapenv:Header>
    <LimitInfoHeader>
      <limitInfo>
        <current>5</current>
        <limit>100000</limit>
        <type>API REQUESTS</type>
      </limitInfo>
    </soapenv:Header>
    <soapenv:Body>
      <queryResponse>
        <result xsi:type="QueryResult">
          <done>true</done>
          <queryLocator xsi:nil="true"/>
          <records xsi:type="sf:sObject">
            <sf:type>dev_ns__Merchandise__c</sf:type>
            <sf:Id>a00D0000008pQSNIA2</sf:Id>
            <sf:dev_ns__Description__c>Phone Case for iPhone
              4/4S</sf:dev_ns__Description__c>
            <sf:dev_ns__Price__c>16.99</sf:dev_ns__Price__c>
            <sf:dev_ns__Stock_Price__c>12.99</sf:dev_ns__Stock_Price__c>
            <sf:dev_ns__Total_Inventory__c>108.0</sf:dev_ns__Total_Inventory__c>
            <sf:Id>a00D0000008pQSNIA2</sf:Id>
          </records>
          <size>1</size>
        </result>
      </queryResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

LocaleOptions

Specifies the language of the labels returned.

API Calls

`describeSObject()`, `describeSObjects()`, `describeDataCategoryGroups()`,
`describeDataCategoryGroupStructures()`

Fields

Element Name	Type	Description
language	string	Specifies the language of the labels returned. The value must be a valid user locale (language and country), such as <code>de_DE</code> or <code>en_GB</code> . For more information on locales, see the Language field on the <code>CategoryNodeLocalization</code> object.

Sample Code—Java

This sample sets the `LocaleOptions` header to the locale of the logged-in user, and then performs a describe on `Account`.

```
public void localeOptionsExample() {
    try {
        connection.setLocaleOptions("en_US");
        connection.describeSObject("Account");
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

LoginScopeHeader

Specifies your organization ID so that you can authenticate Self-Service users for your organization using the existing `login()`.



Note: Starting with Spring '12, the Self-Service portal isn't available for new Salesforce orgs. Existing orgs continue to have access to the Self-Service portal.

API Calls

`login()`

Fields

Element Name	Type	Description
organizationId	ID	The ID of the organization against which you authenticate Self-Service users.
portalId	ID	Specify only if user is a Customer Portal user. The ID of the portal for this organization. The ID is available in the Salesforce user interface: <ul style="list-style-type: none"> From Setup, enter <i>Customer Portal Settings</i> in the Quick Find box, then select Customer Portal Settings Select a Customer Portal name, and on the Customer Portal detail page, the URL of the Customer Portal displays. The Portal ID is in the URL.

Sample Code—C#

This sample shows how to use the `LoginScopeHeader`. It sets the organization ID and the portal ID for a Customer Portal user. It also sets the `CallOptions` header. It then logs the specified user in.

```

/// Demonstrates how to set the LoginScopeHeader values.
public void LoginScopeHeaderSample ()
{
    // Web Reference to the imported Partner WSDL.
    APISamples.partner.SforceService partnerBinding;

    string username = "USERNAME";
    string password = "PASSWORD";

    // The real Client ID will be an API Token provided by Salesforce
    // to partner applications following a security review. For more details,
    // see the Security Review FAQ in the online help.
    string clientId = "SampleCaseSensitiveToken/100";

    partnerBinding = new SforceService();
    partnerBinding.CallOptionsValue = new CallOptions();
    partnerBinding.CallOptionsValue.client = clientId;

    // To authenticate Self-Service users, we need to set the OrganizationId
    // in the LoginScopeHeader.
    string orgId = "00ID0000OrgFoo";
    partnerBinding.LoginScopeHeaderValue = new LoginScopeHeader();
    partnerBinding.LoginScopeHeaderValue.organizationId = orgId;
    // Specify the Portal ID if the user is a Customer Portal user.
    string portalId = "00ID0000FooPtl";
    partnerBinding.LoginScopeHeaderValue.portalId = portalId;

    try
    {
        APISamples.partner.LoginResult lr =
            partnerBinding.login(username, password);
    }
}

```

```

catch (SoapException e)
{
    Console.WriteLine(e.Code);
    Console.WriteLine(e.Message);
}
}

```

MruHeader

In API version 7.0 and later, the `create()`, `update()`, and `upsert()` calls do not update the list of most recently used (MRU) items in the Recent Items section of the sidebar in the Salesforce user interface unless this header is used. Be advised that using this header to update the Recent Items list may negatively impact performance.

API Calls

`create()`, `merge()`, `query()`, `retrieve()`, `update()`, `upsert()`

Fields

Element Name	Type	Description
updateMru	boolean	<p>Indicates whether to update the list of most recently used items (<code>true</code>) or not (<code>false</code>).</p> <p>For <code>retrieve()</code>, if the result has only one row, the MRU is updated to the ID of the retrieve result.</p> <p>For <code>query()</code>, if the result has only one row and the ID field is selected, the MRU is updated to the ID of the query result.</p>

Sample Code—Java

This sample turns on the MRU list update option by setting the `MruHeader` to `true`. Next, it creates an account.

```

public void mruHeaderSample() {
    connection.setMruHeader(true);
    Account account = new Account();
    account.setName("This will be in the MRU");
    try {
        SaveResult[] sr = connection.create(new SObject[]{account});
        System.out.println("ID of account added to MRU: " +
            sr[0].getId());
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
}

```


OwnerChangeOptions

Represents actions that can be performed when a record's owner is changed. Available with these options in API version 35.0 and later.

API Calls

[update\(\)](#), [upsert\(\)](#)

Fields

Element Name	Type	Description
options	OwnerChangeOption[]	Represents a flag for a specific action performed when changing a record owner through an update or upsert call.

OwnerChangeOption Fields

Element Name	Type	Description
execute	boolean	If <code>true</code> , the action represented by the <code>type</code> field is performed. If <code>false</code> , the action represented by the <code>type</code> field is skipped.

type	enum of a string	Represents the action performed or skipped, according to the given value for the execute field, when changing a record owner during an update or upsert call. The following types can be used.
------	------------------	--

EnforceNewOwnerHasReadAccess

If `true`, the record's new owner must have at least read access on the record. Available in API version 36.0 and later.

KeepAccountTeam

If `true`, the account team is kept with the account when the account owner is changed. If `false`, the account team is deleted. Default is `false`. This action only applies to team members added by a Salesforce admin, the account owner, or someone higher in the role hierarchy. Team members added by users with group-based access are removed even if `true`. Available for accounts in API version 45.0 and later.

KeepSalesTeam

If `true`, the opportunity team is kept with the opportunity when the account owner is changed. If `false`, the opportunity team is deleted. Default is `false`. Available for opportunities in API version 45.0 and later.

KeepSalesTeamGrantCurrentOwnerReadWriteAccess

If `true`, the opportunity's previous owner retains read/write access after the owner is changed. Default is `false`. Can be `true` only when `KeepSalesTeam` is `true`. Available for opportunities in API version 44.0 and later.

Element Name	Type	Description
		<p>SendEmail</p> <p>If <code>true</code>, an email notification is sent to the new owner. Default is <code>false</code>.</p>
		<p>TransferAllOwnedCases</p> <p>If <code>true</code>, all cases (open and closed) owned by the account owner are transferred to the new owner. Default is <code>false</code>. When <code>TransferAllOwnedCases</code> is <code>true</code>, <code>TransferOwnedOpenCases</code> must also be true. Available for accounts in API version 45.0 and later.</p>
		<p>TransferArticleOwnedPublishedVersion</p> <p>If <code>true</code> and the record is a Knowledge article, the article owner's published version for the language of the current draft is transferred to the new owner, in addition to the current draft.</p>
		<p>TransferArticleOwnedArchivedVersions</p> <p>If <code>true</code> and the record is a Knowledge article, the article owner's archived versions for the language of the current draft are transferred to the new owner, in addition to the current draft.</p>
		<p>TransferArticleAllVersions</p> <p>If <code>true</code> and the record is a Knowledge article, all published and archived versions owned by anyone for the language of the current draft are transferred to the new owner, in addition to the current draft.</p>
		<p>TransferContacts</p> <p>If <code>true</code> and the record is a business account, contacts associated with the account are transferred to the new owner.</p>
		<p>TransferContracts</p> <p>If <code>true</code> and the record is an account, contracts associated with the account and owned by the account owner are transferred to the new owner.</p>
		<p>TransferNotesAndAttachments</p> <p>If <code>true</code>, the record's notes, attachments, and Google Docs are transferred to the new record owner. If <code>false</code>, the original record owner retains ownership.</p>
		<p>TransferOpenActivities</p> <p>If <code>true</code>, the record's open activities are transferred to the new owner.</p>
		<p>TransferOrders</p> <p>If <code>true</code> and the record is an account, the draft standalone orders associated with the account and draft orders associated with transferred contracts owned by the account owner are transferred to the new owner.</p>
		<p>TransferOthersOpenOpportunities</p> <p>If <code>true</code> and the record is an account, open opportunities associated with the account and not owned by the current owner are transferred to the new owner. When this option is executed, <code>TransferOwnedOpenOpportunities</code> must be set to execute. Default is <code>false</code>.</p>
		<p>TransferOwnedClosedOpportunities</p> <p>If <code>true</code> and the record is an account, closed opportunities owned by the account owner are transferred to the new owner. Default is <code>false</code>. Available for API version 45.0 and later.</p>

Element Name	Type	Description
		<p>TransferOwnedOpenCases</p> <p>If <code>true</code> and the record is an account, open cases owned by the account owner are transferred to the new owner. Default is <code>false</code>. Available for API version 45.0 and later.</p>
		<p>TransferOwnedOpenOpportunities</p> <p>If <code>true</code> and the record is an account, open opportunities associated with the account and owned by the account owner are transferred to the new owner.</p>

Usage

When changing the owners of multiple accounts, all accounts must have the same old owner and the same new owner. To change ownership of accounts with different owners, use separate API requests.

Sample Code—Java

This sample creates an account, a note, an opportunity, and task for the account. It sets the owner change options so that the note, opportunity, and task are transferred to the new owner along with the account.

```
public void ownerChangeOptionsHeaderSample() {

    // Create account. Accounts don't transfer activities, notes, or attachments by default

    Account account = new Account();
    account.setName("Account");
    com.sforce.soap.enterprise.SaveResult[] sr = connection.create(new
com.sforce.soap.enterprise.sobject.SObject[] { account } );
    String accountId = null;

    if(sr[0].isSuccess()) {
        System.out.println("Successfully saved the account");
        accountId = sr[0].getId();

        // Create a note, a task, and an opportunity for the account

        Note note = new Note();
        note.setTitle("Note Title");
        note.setBody("Note Body");
        note.setParentId(accountId);

        Task task = new Task();
        task.setWhatId(accountId);

        Opportunity opportunity = new Opportunity();
        opportunity.setName("Opportunity");
        opportunity.setStageName("Prospecting");
        Calendar dt = connection.getServerTimestamp().getTimestamp();
```

```

dt.add(Calendar.DAY_OF_MONTH, 7);
opportunity.setCloseDate(dt);
opportunity.setAccountId(accountId);

sr = connection.create(new com.sforce.soap.enterprise.subject.SObject[] { note,
task, opportunity } );

if(sr[0].isSuccess()) {
    System.out.println("Successfully saved the note, task, and opportunity");

    com.sforce.soap.enterprise.QueryResult qr = connection.query("SELECT Id FROM
User WHERE FirstName = 'Jane' AND LastName = 'Doe'");
    String newOwnerId = qr.getRecords()[0].getId();
    account.setId(accountId);
    account.setOwnerId(newOwnerId);

    // Set owner change options so account's child note, task, and opportunity
transfer to new owner
    OwnerChangeOption opt1 = new OwnerChangeOption();
    opt1.setExecute(true);
    opt1.setType(OwnerChangeOptionType.TransferOwnedOpenOpportunities); // Transfer
Open opportunities owned by the account's owner

    OwnerChangeOption opt2 = new OwnerChangeOption();
    opt2.setExecute(true);
    opt2.setType(OwnerChangeOptionType.TransferOpenActivities);

    OwnerChangeOption opt3 = new OwnerChangeOption();
    opt3.setExecute(true);
    opt3.setType(OwnerChangeOptionType.TransferNotesAndAttachments);

    connection.setOwnerChangeOptions(new OwnerChangeOption[] {opt1, opt2, opt3});
    connection.update(new com.sforce.soap.enterprise.subject.SObject[] { account }
);

    // The account's note, task, and opportunity should be transferred to the new
owner.
}

} else {
    System.out.println("Account save failed: " + sr[0].getErrors().toString());
}
}

```

PackageVersionHeader

Specifies the package version for each installed managed package.

A managed package can have several versions with different content and behavior. This header allows you to specify the version used for each package referenced by your API client.

If a package version isn't specified, the API client uses the version of the package specified in Setup. From Setup, enter *API* in the **Quick Find** box, select **API**, and then click **Configure Enterprise Package Version Settings** under **Enterprise Package Version Settings**.

This header is available in API version 16.0 and later.

Associated API Calls

`convertLead()`, `create()`, `delete()`, `describeGlobal()`, `describeLayout()`, `describeSObject()`, `describeSObjects()`, `describeSoftphoneLayout()`, `describeTabs()`, `merge()`, `process()`, `query()`, `retrieve()`, `search()`, `undelete()`, `update()`, `upsert()`

Fields

Element Name	Type	Description
<code>packageVersions</code>	<code>PackageVersion[]</code>	A list of package versions for installed managed packages referenced by your API client.

PackageVersion

Specifies a version of an installed managed package. A package version is *majorNumber.minorNumber*, for example `2.1`.

Fields

Field	Type	Description
<code>majorNumber</code>	<code>int</code>	The major version number of a package version.
<code>minorNumber</code>	<code>int</code>	The minor version number of a package version.
<code>namespace</code>	<code>string</code>	The unique namespace of the managed package.

Sample Code—Java

This sample sets the package version for one installed package in the `PackageVersionHeader`. Next, it executes the code passed into this method via the `executeAnonymous` Apex method.

```
public void PackageVersionHeaderSample(String code) throws Exception
{
    _PackageVersionHeader pvh = new _PackageVersionHeader();
    PackageVersion pv = new PackageVersion();
    pv.setNamespace("installedPackageNamespaceHere");
    pv.setMajorNumber(1);
    pv.setMinorNumber(0);
    // In this case, we are only referencing one installed package.
    PackageVersion[] pvs = new PackageVersion[]{pv};
    pvh.setPackageVersions(pvs);

    apexBinding.setHeader(new SforceServiceLocator().getServiceName().getNamespaceURI(),
        "PackageVersionHeader", pvh);
}
```

```

// Execute the code passed into the method.
ExecuteAnonymousResult r = apexBinding.executeAnonymous(code);
if (r.isSuccess()) {
    System.out.println("Code executed successfully");
}
else {
    System.out.println("Exception message: " + r.getExceptionMessage());
    System.out.println("Exception stack trace: " + r.getExceptionStackTrace());
}
}

```

QueryOptions

Specifies the preferred batch size for queries. The system sometimes creates batches that are larger or smaller than the specified size to maximize performance.

Associated API Calls

[query\(\)](#), [queryMore\(\)](#), [retrieve\(\)](#)

Fields

Element Name	Type	Description
batchSize	int	<p>The batch size for the number of records returned in a <code>query()</code> or <code>queryMore()</code> call. Child objects count toward the number of records for the batch size. For example, in relationship queries, multiple child objects are returned per parent row returned.</p> <p>The default and the maximum are 2,000; the minimum is 200. There is no guarantee that the requested batch size is the actual batch size. To maximize performance, the number of results returned can vary based on the size and complexity of the records..</p>

Sample Code

For code examples, see [Change the Batch Size in Queries in the Salesforce SOQL and SOSL Reference Guide](#).

SessionHeader

Specifies the session ID returned from the login server after a successful `login()`. This session ID is used in all subsequent calls.

In version 12.0 and later, include the API namespace in the SOAP message associated with this header. The namespace is defined in the enterprise or partner WSDL.

API Calls

All calls, including utility calls.


Fields

Element Name	Type	Description
<code>sessionId</code>	string	Session ID returned by the <code>login()</code> call to be used for subsequent call authentication.

Sample Code

See the examples provided for `login()`.

UserTerritoryDeleteHeader

 **Note:** The original territory management feature is now unavailable. For more information, see [The Original Territory Management Module Will Be Retired in the Summer '21 Release](#). The information in this topic applies to the original territory management feature only, and not to Enterprise Territory Management.

Specify a user to whom open opportunities are assigned when the current owner is removed from a territory. If this header is not used or the value of its element is null, the opportunities are transferred to the forecast manager in the territory above, if one exists. If one does not exist, the user being removed from the territory keeps the opportunities.

API Calls

`delete()`

Fields

Element Name	Type	Description
<code>transferToUserId</code>	ID	The ID of the user to whom open opportunities in that user's territory will be assigned when an opportunity's owner (user) is removed from a territory.

USING THE API WITH SALESFORCE FEATURES

CHAPTER 13 Implementation Considerations

In this chapter ...

- Choosing a User for an Integration
- Login Server URL
- Log In to the Login Server
- Typical API Call Sequence
- Salesforce Sandbox
- Multiple Instances of Salesforce Database Servers
- Content Type Requirement
- API Usage Metering
- Compression
- HTTP Persistent Connections
- HTTP Chunking
- Internationalization and Character Sets
- XML Compliance
- .NET, Non-String Fields, and the Enterprise WSDL

Before you build an integration app or other client app, consider the data management, use limits, and communication issues explained in this section.

Choosing a User for an Integration

When your client app connects to the API, it must first log in. You must specify a user to log in to Salesforce when calling `login()`. Client apps run with the permissions and sharing of the logged-in user. Use the following sections to help decide how to configure a user for your client app.

Permissions

As an org's Salesforce admin, you control which features and views are available to users by configuring profiles and permission sets and assigning users to them. To access the API to issue calls and receive the call results, a user must have the API Enabled permission. Client apps can query or update only those objects and fields to which they have access via the permissions of the logged-in user.

If the client application logs in as a user who has access to data via a sharing rule, then the API must issue an extra query to check access. To avoid this, log in as a user with the "Modify All Data" permission. This can speed up the call response time. If providing the Modify All Data permission is too permissive for a particular user, consider using the Modify All object-level permission to restrict data access on an object basis. For more information, see [Factors that Affect Data Access](#).

Limits

Salesforce limits the number of queries that a user can execute concurrently. A user can have up to 10 query cursors open at a time. If 10 `QueryLocator` cursors are open when a client application, logged in as the same user, attempts to open a new one, then the oldest of the 10 cursors is released. If the client application attempts to open the released query cursor, an error results.

Multiple client apps can log in using the same `username` argument. However, this approach increases your risk of getting errors due to query limits.


If multiple client apps are logged in with the same user, they all share the same session. If one of the client apps calls `logout()`, it invalidates the session for all the client apps. Using a different user for each client app makes it easier to avoid these limits.

 **Note:** In addition to user limits, Salesforce limits for API requests for each org. For more information, see [API Usage Metering](#).

Login Server URL

SOAP API provides a single login server. You can log in to any org from a single entry point without hard coding the instance. To access an org via the API, first authenticate the session by sending a `login()` request to the login server at one of the following URLs, depending on your choice of WSDL.

- `https://MyDomainName.my.salesforce.com/services/Soap/c/55.0` or `https://login.salesforce.com/services/Soap/c/55.0` (enterprise WSDL)
- `https://MyDomainName.my.salesforce.com/services/Soap/u/55.0` or `https://login.salesforce.com/services/Soap/u/55.0` (partner WSDL)

 **Note:** HTTPS is required. `login()` requests that use HTTP, for example, `https://login.salesforce.com/services/Soap/u/54.0`, aren't supported.

All subsequent calls to the server during the session should be made to the URL returned in the `login()` response, which points to the server instance for your org.

Log In to the Login Server

Before invoking any other calls, a client app must first invoke the `login()` call to establish a session with the login server. It then sets the returned server URL as the target server for subsequent API requests and sets the returned session ID in the SOAP header to provide server authorization for subsequent API requests. Salesforce checks the IP address from which the client app is logging in and blocks logins from unknown IP addresses. For more information, see `login()` and [Step 4: Walk Through the Sample Code](#).

If the API blocks the login, Salesforce returns a login fault. To log in, the user must add the security token at the end of the user's password. For example, if a user's password is `mypassword` and the security token is `XXXXXXXXXX`, the user enters `mypasswordXXXXXXXXXX`. Users get their security token by changing their password or resetting their security token from the Salesforce user interface. When users change their password or reset their security token, Salesforce sends a new security token to the email address on the user's Salesforce record. The security token is valid until the user resets the security token, or changes the password, or you reset the user's password. When the security token is invalid, the user must repeat the login process. To avoid another log in, add the client's IP address to the org's list of trusted IP addresses. For more information, see [Security Token](#).

When you are logged in, you can issue API calls. For each operation, client apps submit a synchronous request to the API, await the response, and then process the results. The API commits changed data automatically.

API calls:

- [Core Calls](#)
- [Describe Calls](#)
- [Utility Calls](#)

Typical API Call Sequence

For each call, your client app typically:

1. Prepares the request by defining request parameters, if applicable.
2. Invokes the call, which passes the request with its parameters to the Lightning Platform Web Service for processing.
3. Receives the response from the API.
4. Handles the response, either by processing the returned data (for a successful invocation) or by handling the error (for a failed invocation).

Salesforce Sandbox

Professional, Enterprise, Unlimited, and Performance Edition customers have access to the Salesforce Sandbox, which is a testing environment that offers a full or partial copy of your Salesforce org's live production data. For more information, visit the Salesforce Community website at www.salesforce.com/community or see [Sandbox Types and Templates](#) in the Salesforce Help.

To access your org's sandbox via the API, use the following URLs to make login requests.

- <https://test.salesforce.com/services/Soap/c/55.0> (enterprise WSDL)
- <https://test.salesforce.com/services/Soap/u/55.0> (partner WSDL)

Multiple Instances of Salesforce Database Servers

Although orgs are generally allocated by geographic regions, an org may be on any instance.

Content Type Requirement

In the API version 7.0 and later, all requests must contain a correct content type HTTP header, for example: `Content-Type: text/xml; charset=utf-8`. Earlier versions of the API do not enforce this requirement.

API Usage Metering

To maintain optimum performance and ensure that the Lightning Platform API is available to all our customers, Salesforce balances transaction loads by imposing two types of limits:

- Concurrent API Request Limits
- Total API Request Allocations

When a call exceeds a request limit, an error is returned.

Concurrent API Request Limits

The following table lists the limits for various types of orgs for concurrent inbound requests (calls) with a duration of 20 seconds or longer.

Org Type	Limit
Developer Edition and Trial orgs	5
Production orgs and Sandboxes	25

Total API Request Allocations

The following table lists the limits for the total inbound API requests (calls) per 24-hour period for an org.

Salesforce Edition	API Calls Per License Type Per 24-Hour Period	Total Calls Per 24-Hour Period
Developer Edition	N/A	15,000
<ul style="list-style-type: none"> • Enterprise Edition • Professional Edition with API access enabled 	<ul style="list-style-type: none"> • Salesforce: 1,000 • Salesforce Platform: 1,000 • Lightning Platform - One App: 200 • Customer Community: 0 • Customer Community Login: 0 • Customer Community Plus: 200 • Customer Community Plus Login: 10 • External Identity 25,000 SKU: 70,000 • External Identity 250,000 SKU: 750,000 • External Identity 1,000,000 SKU: 4,000,000 • Partner Community: 200 	100,000 + (number of licenses x calls per license type) + purchased API Call Add-Ons

Salesforce Edition	API Calls Per License Type Per 24-Hour Period	Total Calls Per 24-Hour Period
	<ul style="list-style-type: none"> Partner Community Login: 10 Lightning Platform Starter: 200 per member for Enterprise Edition orgs Lightning Platform Plus: 1000 per member for Enterprise Edition orgs 	
<ul style="list-style-type: none"> Unlimited Edition Performance Edition 	<ul style="list-style-type: none"> Salesforce: 5,000 Salesforce Platform: 5,000 Lightning Platform - One App: 200 Customer Community: 0 Customer Community Login: 0 Customer Community Plus: 200 Customer Community Plus Login: 10 External Identity 25,000 SKU: 70,000 External Identity 250,000 SKU, 750,000 External Identity 1,000,000 SKU: 4,000,000 Partner Community: 200 Partner Community Login: 10 Lightning Platform Starter: 200 per member for Unlimited and Performance Edition orgs Lightning Platform Plus: 5,000 per member for Unlimited and Performance Edition orgs 	100,000 + (number of licenses x calls per license type) + purchased API Call Add-Ons
Sandbox	N/A	5,000,000

For Experience Cloud limits, see [Experience Cloud User Licenses](#).

 **Note:** Load, performance, and other system issues can prevent you from using your entire allocation of calls in a 24-hour period.

APIs that count toward this allocation include the Lightning Platform REST API, the Lightning Platform SOAP API, Bulk API, and Bulk API 2.0. API calls issued by certain Salesforce connected apps (for example, the Salesforce mobile app) don't count. To determine which APIs affect the allocation, see [Monitoring Your API Usage](#).

Calls that include DebuggingHeader have a separate allocation limit of 1,000 calls per 24-hour period. These calls can continue to be made after the total request limit for an org is reached.

Limits and allocations are enforced against the aggregate of all API calls made to the org in a 24-hour period. Limits and allocations are on a per-user basis.

Monitoring Your API Usage

To better monitor your org's API usage and limits, you can use these resources:


- The API Usage section of the System Overview page in Setup.
- The API Requests, Last 24 Hours item in the Organization Detail section of the System Overview page in Setup.
- The API Request Limit per Month usage-based entitlement, which shows you your org's API calls aggregated over 30 days. This can be found on the Company Information page in Setup.
- Information returned in the `Sforce-Limit-Info` response header for REST APIs.
- Information returned in the response body (in `<type>API REQUESTS</type>`) for SOAP APIs.
- The `/limits` call in the Lightning Platform REST API.

You can configure your org so that email is sent to a designated user when the number of API requests has exceeded a specified percentage of the amount allotted. Perform this configuration from Setup by entering *API Usage Notifications* in the Quick Find box and then selecting **API Usage Notifications**.

See also the [Learn About Daily Rate Limits](#) section in the App Development Without Limits Trailhead module.

What Happens If You Reach or Exceed Your API Request Limit

If your org reaches or exceeds its daily API request limit, Salesforce still allows the operations to proceed by a certain amount, if possible. This helps avoid blocking your workflows during unexpected spikes in workloads and occasional peak periods. A hard cap is in place to safeguard platform resources and prevent API requests from exceeding the daily limit unimpeded.

 **Note:** The ability to go over your normal daily limit is always subject to restrictions to protect the overall health of the Salesforce instance that hosts your org. (You can monitor the health of your instance on [Salesforce Trust](#).)

This ability is designed to be used occasionally to help avoid interruptions in your workflow. Don't rely on it on an ongoing basis. To increase your allocation, contact your Salesforce account representative.

This ability only applies to paid orgs in active status. It does not apply to trial orgs, Developer Edition, or sandboxes.

API request activity is aggregated into 30 day periods, starting with your contract start date, and includes calls that exceed the org's entitled limit.

Increasing Total API Request Allocations

The calculation of the API request amounts based on user licenses is designed to allow sufficient capacity for your org based on your number of users. If you need a higher amount and you don't want to purchase extra user licenses or upgrade to Performance Edition, you can purchase extra API calls. For information, contact your account representative.

Before you purchase more API calls, perform due diligence of your API usage. You can optimize a client application, whether it's your own enterprise application or partner application, to use fewer API calls and still accomplish the same work. If you use a partner product, consult with the vendor to verify that the product makes optimal use of the API. A product that makes inefficient use of the API incurs unnecessary costs for your company. Use REST API [composite resources](#) to improve your application's performance by minimizing the number of round-trips between the client and server.

Example API Usage Metering Calculations

The following examples illustrate API usage metering calculations for several scenarios.

- For an Enterprise Edition org with 15 Salesforce licenses, the request limit is 115,000 requests (100,000 plus 15 licenses x 1,000 calls).

- For a Developer Edition org that made 14,500 calls at 5:00 AM Wednesday, 499 calls at 11:00 PM Wednesday, only one more call can successfully be made until 5:00 AM Thursday.

Length of Stored Third-Party Refresh and Access Tokens

Salesforce stores third-party access and refresh tokens of up to 10,000 characters in length.

Compression

The API allows the use of compression on the request and the response, using the standards defined by the HTTP 1.1 specification. This is automatically supported by some SOAP/WSDL clients, and can be manually added to others. Visit <https://developer.salesforce.com/page/Tools> for more information on particular clients.

Compression is not used unless the client specifically indicates that it supports compression. For better performance, we suggest that clients accept and support compression as defined by the HTTP 1.1 specification.

To indicate that the client supports compression, you should include the HTTP header "Accept-Encoding: gzip, deflate" or a similar heading. The API compresses the response if the client properly specifies this header. The response includes the header "Content-Encoding: deflate" or "Content-Encoding: gzip," as appropriate. You can also compress any request by including a "Content-Encoding: deflate" or "gzip" header.

Most clients are partially constrained by their network connection, even on a corporate LAN. The API allows the use of compression to improve performance. Almost all clients can benefit from response compression, and many clients may benefit from compression of requests as well. The API supports deflate and gzip compression according the HTTP 1.1 specification.


Response Compression

The API can optionally compress responses. Responses are compressed only if the client sends an Accept-Encoding header with either gzip or deflate compression specified. The API is not required to compress the response even if you have specified Accept-Encoding, but it normally does. If the API compresses the response, it also specifies a Content-Encoding header with the name of the compression algorithm used, either gzip or deflate.

Request Compression

Clients can also compress requests. The API decompresses any requests before processing. The client must send up a Content-Encoding HTTP header with the name of the appropriate compression algorithm. For more information, see:

- Content-Encoding at: www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11
- Accept-Encoding at: www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.3
- Content Codings at: www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.5

 **Note:** To implement request SOAP compression in a Java client with WSC (Web Service Connector), call `setCompression()` on the `Config` you use to instantiate a `Connection` object with. For an example, see [login\(\) sample](#) on page 157 code.

HTTP Persistent Connections

Most clients achieve better performance if they use HTTP 1.1 persistent connection to reuse the socket connection for multiple requests. Persistent connections are normally handled by your SOAP/WSDL client automatically. For more details, see the HTTP 1.1 specification at:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.1>

HTTP Chunking

Clients that use HTTP 1.1 may receive chunked responses. Chunking is normally handled by your SOAP/WSDL client automatically.

Internationalization and Character Sets

The API supports either full Unicode characters or ISO-8859-1 characters. The character set depends on the Salesforce instance that your org uses. If your org logs in to `ssl.salesforce.com`, your encoding is ISO-8859-1. All other instances use UTF-8. To determine the character set, call `describeGlobal()` and inspect the `encoding` value returned in `DescribeGlobalResult`.

If your org uses ISO-8859-1 encoding, all data sent to the API must be encoded in ISO-8859-1. Characters outside the valid ISO-8859-1 range might be truncated or cause an error.

 **Note:** The API response is encoded in the character set used by your org (UTF-8 or ISO-8859-1). Either way, the encoded data is usually handled for you by the SOAP client.

XML Compliance

The API is based on XML, which requires all documents to be well formed. Part of that requirement is that certain Unicode characters are not allowed in an XML document, even in an escaped form, and that others must be encoded according to their location. Normally this is handled for you by any standard SOAP or XML client. Clients must be able to parse any normal XML escape sequence, and must not pass up invalid XML characters.

Some characters, as mentioned, are illegal even if they are escaped. The illegal characters include unpaired Unicode surrogates and a few other Unicode characters. All are seldom-used control characters that are usually not important in any data, and tend to cause problems with many programs. Although they are not allowed in XML documents, they are allowed in HTML documents and may be present in Salesforce data. The illegal characters will be stripped from any API response.

Illegal characters:

- 0xFFFFE
- 0xFFFF
- Control characters 0x0 - 0x19, except the following characters, which are legal: 0x9, 0xA, 0xD, tab, newline, and carriage return)
- 0xD800 - 0xDFFF, unless they're used to form a surrogate pair

.NET, Non-String Fields, and the Enterprise WSDL

If you use .NET with the enterprise WSDL, .NET generates a Boolean field for each non-string field. For example, if you have a date value in `MyDateField__c`, .NET generates a Boolean field called `MyDateField__cSpecified`.

The generated field value is `false` by default. If a Specified field value is `false`, then the values in the corresponding original field are not be included in the SOAP message. For example, before the values in the currency field `annualRevenue` can be included in a SOAP message generated by your client app, the value of `annualRevenueSpecified` must be set to `true`.

```
account.annualRevenue = 10000;
account.annualRevenueSpecified = true;
```

CHAPTER 14 Objects and SOAP API Calls and Headers for Apex

These Salesforce Objects and SOAP API calls and headers are available by default for Apex. For information on all other SOAP API calls, including those that can be used to extend or implement any existing Apex IDEs, contact your Salesforce representative.

Apex class methods can be exposed as custom SOAP Web service calls. This allows an external application to invoke an Apex Web service to perform an action in Salesforce. Use the `webservice` keyword to define these methods. For more information, see [Considerations for Using the `webservice` Keyword](#).

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 55.0, use endpoint 55.0:

```
https://MyDomain.salesforce.com/services/Soap/s/55.0
```

These Salesforce objects are available for Apex.

- [ApexTestQueueItem](#)
- [ApexTestResult](#)
- [ApexTestResultLimits](#)
- [ApexTestRunResult](#)

Use these SOAP API calls to deploy your Apex.

- `compileAndTest()`
- `compileClasses()`
- `compileTriggers()`
- `executeAnonymous()`
- `runTests()`

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.

These SOAP headers are available in SOAP API calls for Apex.

- [DebuggingHeader](#)
- [PackageVersionHeader](#)

SEE ALSO:

[Apex Developer Guide](#)

CHAPTER 15 Outbound Messaging

In this chapter ...

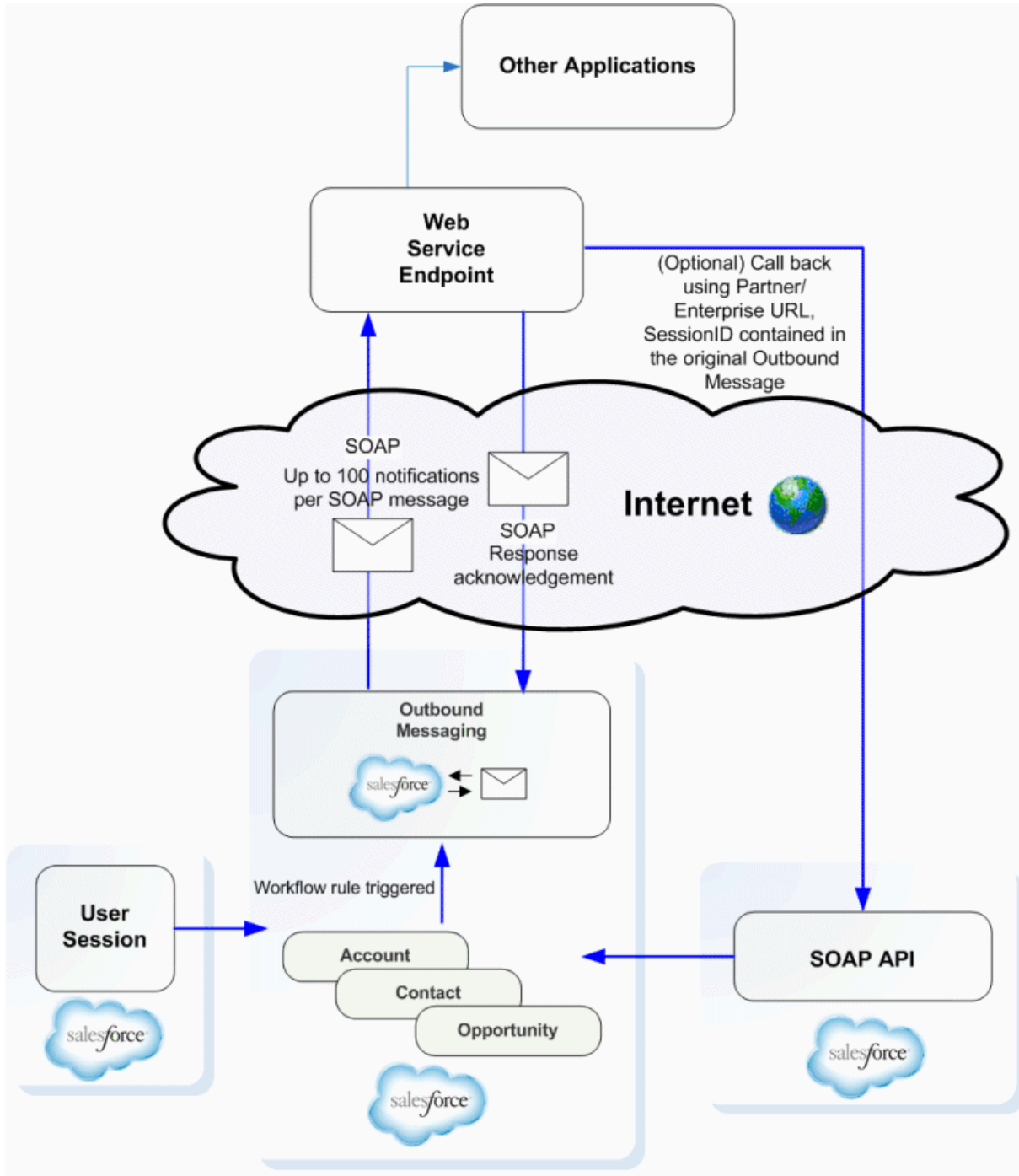
- [Understanding Outbound Messaging](#)
- [Understanding Notifications](#)
- [Setting Up Outbound Messaging](#)
- [Considerations for Security](#)
- [Understanding the Outbound Messaging WSDL](#)
- [Building a Listener](#)

Outbound messaging allows you to specify that changes to fields within Salesforce can cause messages with field values to be sent to designated external servers.

Outbound messaging is part of the workflow rule functionality in Salesforce. Workflow rules watch for specific kinds of field changes and trigger automatic Salesforce actions, such as sending email alerts, creating task records, or sending an outbound message.

Understanding Outbound Messaging

Outbound messaging uses the `notifications ()` call to send SOAP messages over HTTP(S) to a designated endpoint when triggered by a workflow rule.



After you set up outbound messaging, when a triggering event occurs, a message is sent to the specified endpoint URL. The message contains the fields specified when you created the outbound message. After the endpoint URL receives the message, it can take the information from the message and process it. To do that, you must examine the outbound messaging WSDL.


Understanding Notifications

A single SOAP message can include up to 100 notifications. Each notification contains the object ID and a reference to the associated [sObject](#) data. If the information in the object changes after the notification is queued but before it's sent, only the latest data is delivered and not the intermediate changes.

If you issue multiple discrete calls, the calls are sometimes batched together into one or more SOAP messages.

Messages are queued locally. A separate background process performs the actual sending, to preserve message reliability:

- If the endpoint is unavailable, messages stay in the queue until sent successfully, or until they're 24 hours old. After 24 hours, messages are dropped from the queue.
- If a message can't be delivered, the interval between retries increases exponentially, up to a maximum of two hours between retries.
- Messages are retried independent of their order in the queue. As a result, messages can be delivered out of order.
- You can't build an audit trail using outbound messaging. While each message is usually delivered once, it can sometimes be delivered more than once. If delivery can't be done within 24 hours, a message isn't delivered at all. Finally, if the source object changes after a notification is queued but before it's sent, the endpoint only receives the latest data, not any intermediate changes.
- Because a message can sometimes be delivered more than once, check the notification IDs in the notifications delivered to your listener client before processing.

 **Note:** Instead of polling, which was required in previous releases, you can now use outbound messaging to trigger execution logic when Salesforce raises an event. In previous versions of the API, client applications had to poll Salesforce to find out if relevant changes had occurred. Because most changes eventually trigger a workflow if a rule exists for it, you can use the workflow rule to trigger actions based on Salesforce events.

The metadata needed for outbound messaging, including the definition of the `notifications()` call, which sends the outbound SOAP message to an external service, is in a separate WSDL. The WSDL is created and available from the Salesforce user interface after a workflow rule has been associated with an outbound message. The WSDL is bound to the outbound message and contains the instructions about how to reach the endpoint service and what data is sent to it. For more information about setting up outbound messaging, see [Defining Outbound Messaging](#).

Setting Up Outbound Messaging

Before you can use outbound messaging, you must set it up via the Salesforce user interface.

- [Setting Up User Profiles](#)
- [Defining Outbound Messaging](#)
- [Downloading the Salesforce Client Certificate](#)
- [Viewing Outbound Messages](#)
- [Tracking Outbound Message Status](#)

Setting Up User Profiles

It's possible to create circular changes with outbound messaging. For example, if a user is performing integrations that trigger workflow, and the workflow actions trigger account updates, those account updates trigger new workflow, and so on. To prevent these circular changes, you can disable a user's ability to send outbound messages.

Here's another example of a circular change scenario.

1. You configure an outbound message to include a `sessionId` and specify a user in the **User to send as** field. The user doesn't have outbound messaging disabled.
2. A change in a contact record triggers an outbound message from the specified user, with the `sessionId` to your outbound message listener.
3. Your outbound message listener calls the Lightning Platform API and updates the same contact record which triggered the outbound message.
4. The update triggers an outbound message.
5. Your outbound message listener updates the record.
6. The update triggers an outbound message.
7. Your outbound message listener updates the record.

To disable outbound message notifications for a user, deselect **Send Outbound Messages** in the user's [Profile](#). We recommend specifying a single user to respond to outbound messages and disabling this user's ability to send outbound messages.


Defining Outbound Messaging

To define outbound messages, use this procedure in the Salesforce user interface:

1. From Setup, enter *Outbound Messages* in the **Quick Find** box, then select **Outbound Messages**.
2. Click **New Outbound Message**.
3. Choose the object that has the information you want included in the outbound message, and click **Next**.
4. Configure the outbound message.
 - a. Enter a name and description for this outbound message.
 - b. Enter an endpoint URL for the recipient of the message. Salesforce sends a SOAP message to this endpoint.
For security reasons, Salesforce restricts the outbound ports you can 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.
 - c. Select the Salesforce user to use when sending the message by specifying a username in the **User to send as** field. The chosen user controls data visibility for the message that is sent to the endpoint.
 - d. Select **Send Session ID** if you want a `sessionId` to be included in the outbound message. Include the `sessionId` in your message if you intend to make API calls back to Salesforce from your listener. The `sessionId` represents the user defined in the previous step and not the user who triggered the workflow.
 - e. Select the fields you want included in the outbound message and click **Add**.
5. Click **Save**, and review the outbound message detail page:

- The `API Version` field is automatically generated and set to the current API version when the outbound message was created. This API version is used in API calls back to Salesforce using the enterprise or partner WSDL. The `API Version` can only be modified by using the Metadata API.
- Click **Click for WSDL** to view the WSDL associated with this message.

The WSDL is bound to the outbound message and contains the instructions about how to reach the endpoint service and what data is sent to it.


 **Note:** If you don't have these options, your org doesn't have outbound messaging enabled. Contact Salesforce to enable outbound messaging for your org.

Downloading the Salesforce Client Certificate

Your application (endpoint) server's SSL/TLS can be configured to require client certificates (two-way SSL/TLS), in order to validate the identity of the Salesforce server when it takes the role of client to your server. You can download the Salesforce client certificate from the Salesforce application user interface. This certificate is the client certificate that Salesforce sends with each outbound message for authentication.

1. From Setup, enter `API` in the `Quick Find` box, then select **API**.
2. On the API WSDL page, click **Manage API Client Certificate**.
3. On the Certificate and Key Management page, in the API Client Certificate section, click the **API Client Certificate**.
4. On the Certificates page, click **Download Certificate**. The `.crt` file is saved in the download location specified in your browser.

Import the downloaded certificate into your application server, and configure your application server to request the client certificate. The application server then checks that the certificate used in the SSL/TLS handshake matches the one you downloaded.

 **Note:** Your application (endpoint) server must send any intermediate certificates in the certificate chain, and the certificate chain must be in the correct order. The correct order is:

1. Server certificate
2. Intermediate certificate that signed the server certificate if the server certificate wasn't signed directly by a root certificate
3. Intermediate certificate that signed the certificate in step 2
4. Any remaining intermediate certificates

Don't include the root certificate authority certificate. The root certificate isn't sent by your server. Salesforce already has its own list of trusted certificates on file, and a certificate in the chain must be signed by one of those root certificate authority certificates.

Viewing Outbound Messages

To view existing outbound messages, from Setup, enter `Outbound Messages` in the `Quick Find` box, then select **Outbound Messages** in the Salesforce user interface.

- Click **New Outbound Message** to define a new outbound message.
- Click **View Message Delivery Status** to track the status of an outbound message.
- Select an existing outbound message to view details about it or view workflow rules and approval processes that use it.
- Click **Edit** to make changes to an existing outbound message.
- Click **Del** to delete an outbound message.

Tracking Outbound Message Status

To track the status of an outbound message, from Setup, enter *Outbound Messages* in the **Quick Find** box, select **Outbound Messages**, and then click **View Message Delivery Status**. You can perform several tasks on this page.

- View the status of your outbound messages, including the total number of attempted deliveries.
- View the action that triggered the outbound message by clicking any workflow or approval process action ID.
- Click **Retry** to change the **Next Attempt** date to now. This action causes the message delivery to be immediately retried.
- Click **Del** to permanently remove the outbound message from the queue.

Considerations for Security

To use outbound messaging, ensure that no third party can send messages to the endpoint while pretending to be from Salesforce:

- Lock down the client application's listener to accept requests only from Salesforce IP ranges. While this action guarantees that the message came from Salesforce, it doesn't guarantee that another customer isn't pointing to your endpoint and sending messages. For an up-to-date list of Salesforce IP ranges, see <https://help.salesforce.com/articleView?id=000321501&type=1&mode=1>
- Use SSL/TLS. Using SSL/TLS provides confidentiality while data is transported across the internet. Without it, a malicious third party can eavesdrop on your data. This issue is especially important if you pass data with privacy requirements and you pass a `sessionId` with the message. Also, we authenticate the certificate presented on connection, ensure that it is from a valid Certificate Authority, and check that the domain in the certificate matches the one Salesforce is trying to connect. This validation prevents us from communicating with the wrong endpoint.
- When you select **Send Session ID**, only HTTPS is supported for the endpoint URL to ensure secure transmission of the session ID. For managed and unmanaged packages created before Spring '19 with this option but without an HTTPS endpoint, subscribers can still install them. Starting in Spring '19, you can't create packages with insecure outbound message options.
- The `sessionId` included in the outbound message is scoped only for API requests and doesn't apply to UI requests.
- If the configuration of your application (endpoint) server's SSL/TLS allows, validate the identity of the Salesforce server when it takes the role of a client to your server, using the Salesforce client certificate. For instructions to download the certificate, see [Downloading the Salesforce Client Certificate](#).
- The organization `Id` is included in each message. For more information about the `Id` field type, see [ID Field Type](#). In your client application, validate that messages contain your organization `Id`.

Understanding the Outbound Messaging WSDL

The rest of this topic examines relevant sections of the outbound messaging WSDL. Your WSDL can differ, depending on the choices you made when you set up outbound messaging for a particular event on a particular object.

notifications ()

This section defines the `notifications ()` call, which creates an outbound message containing specified fields and values for a particular object or objects, and sends the values to a specified endpoint URL:

```
<schema elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://soap.sforce.com/2005/09/outbound">
  <import namespace="urn:enterprise.soap.sforce.com" />
  <import namespace="urn:subject.enterprise.soap.sforce.com" />
```

```

<element name="notifications">
  <complexType>
    <sequence>
      <element name="OrganizationId" type="ent:ID" />
      <element name="ActionId" type="ent:ID" />
      <element name="SessionId" type="xsd:string" nillable="true" />
      <element name="EnterpriseUrl" type="xsd:string" />
      <element name="PartnerUrl" type="xsd:string" />
      <element name="Notification" maxOccurs="100"
        type="tns:OpportunityNotification" />
    </sequence>
  </complexType>
</element>
</schema>

```

Use this table to understand the elements named in the notifications method definition:

Name	Type	Description
OrganizationId	ID	ID of the organization sending the message.
ActionId	string	The workflow rule (action) that triggers the message.
SessionId	string	Optional, a session ID to be used by endpoint URL client that is responding to the outbound message. It's used by the receiving code to make calls back to Salesforce.
EnterpriseURL	string	URL to use to make API calls back to Salesforce using the enterprise WSDL.
PartnerURL	string	URL to use to make API calls back to Salesforce using the partner WSDL.
Notification	Notification	Defined in the next section, contains the object datatype and its Id, for example OpportunityNotification or ContactNotification.

The Notification datatype is defined in the WSDL. In the following example, a Notification for opportunities is defined, based on the Notification entry of the `notifications()` call definition:

```

<complexType name="OpportunityNotification">
  <sequence>
    <element name="Id" type="ent:ID" />
    <element name="sObject" type="ens:Opportunity" />
  </sequence>
</complexType>

```

Each object element (in our example, opportunities) contains the subset of the fields that you selected when you [created the outbound message](#). Each message Notification also has the object ID. Use the object ID to track redelivery attempts of notifications you've already processed.

notificationsResponse

This element is the schema for sending an acknowledgment (ack) response to Salesforce.

```

<element name="notificationsResponse">
  <complexType>
    <sequence>

```

```

        <element name="Ack" type="xsd:boolean" />
    </sequence>
</complexType>
</element> //This section is the last in the types definition section.

```

You acknowledge all notifications in the message if there's more than one.

Building a Listener

After you've defined an outbound message and configured an outbound messaging endpoint, download the WSDL and create a listener:

1. Right-click **Click for WSDL** and select Save As to save the WSDL to a local directory with an appropriate file name. For example, for an outbound message that deals with leads, you could name the WSDL file `leads.wsdl`.
2. Unlike the enterprise or partner WSDLs, which describe the messages the client sends to Salesforce, this WSDL defines the messages that Salesforce sends to your client application.
3. Most Web services tools generate stub listeners for you, in much the same way as they generate a client stub for the enterprise or partner WSDL. Look for a server-side stub option.

For example, for .NET 2.0:

- a. Run `wsdl.exe /serverInterface leads.wsdl` with .NET 2.0. This command generates `NotificationServiceInterfaces.cs`, which defines the notification interface.
- b. Create a class that implements `NotificationServiceInterfaces.cs`.
- c. You implement your listener by writing a class that implements this interface. There are a number of ways to do this. One simple way is to compile the interface to a DLL first (DLLs must be in the `bin` directory in ASP.NET).

```

mkdir bin
csc /t:library /out:bin\nsi.dll NotificationServiceInterfaces.cs

```

Now write an ASMX-based Web service that implements this interface. For example, in `MyNotificationListener.asmx`:

```

<%@WebService class="MyNotificationListener" language="C#"%>
class MyNotificationListener : INotificationBinding
{
    public notificationsResponse notifications(notifications n)
    {
        notificationsResponse r = new notificationsResponse();
        r.Ack = true;
        return r;
    }
}

```


This example is a simple implementation, actual implementations are more complex.

- d. Deploy the service by creating a new virtual directory in IIS for the directory that contains the `MyNotificationListener.asmx`.
- e. You can now test that the service is deployed by viewing the service page with a browser. For example, if you create a virtual directory `salesforce`, you'd go to `http://localhost/salesforce/MyNotificationListener.asmx`.

The process for other Web service tools is similar. Consult the documentation for your Web service tool.

Your listener must meet these requirements:

- Must be reachable from the public Internet.
- For security reasons, Salesforce restricts the outbound ports you can 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.
- To be valid, the common name (CN) of the certificate must match the domain name for your endpoint's server, and the certificate must be issued by a Certificate Authority trusted by Java 2 Platform, Standard Edition (J2SE) 5.0 (JDK 1.5).
- If your certificate expires, message delivery fails.
-  **Warning:** To avoid an infinite loop of outbound messages that trigger changes that trigger more outbound messages, ensure that the user who updates the objects does **not** have the “Send Outbound Messages” permission.

CHAPTER 16 Data Loading and Integration

In this chapter ...

- [Client Application Design](#)
- [Salesforce Settings](#)
- [Best Practices with Any Data Loader](#)
- [Integration and Single Sign-On](#)

If you need to load large volumes of data (hundreds of thousands to millions of records), there are a number of factors you must consider. Use the topics in this section to become familiar with issues of client application design, organization configuration, and data loader best practices.

Client Application Design

Although the Bulk API 2.0 is the best choice for loading large numbers of records, you can also use the SOAP-based API. There are many ways you can design your application to improve the speed of data loads:

- **Use persistent connections.** Opening a socket takes time, mostly when opening a socket stems from the SSL/TLS negotiation. Without SSL or TLS, the API request would not be secure. Included in the HTTP 1.1 specification is support for reusing sockets among requests (persistent connections) instead of having to re-open a socket per request as in HTTP 1.0. Whether your client supports persistent connections depends on the SOAP stack you are using. By default, .NET uses persistent connections. If you change the configuration to use the Apache http-commons libraries, your client will be compliant with the HTTP 1.1 specification and use persistent connections.

For information about HTTP 1.1, see [HTTP Persistent Connections](#) and <http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.1>.

- **Minimize the number of requests.** There is some processing associated with each request, so to save time your client should batch as many records per request as possible. Set `batchSize` to the limit of 2,000. If that is not the most efficient batch size, the API will change it. For more information about setting batch sizes, see [QueryOptions](#).
- **Minimize the size of the requests.** Your client application should send as many records per request as possible, but it should also send as small a request as possible to reduce network transmission time. To minimize the request size, use compression on both the request and the response. Gzip is the most popular type of compression, and there are multiple posts on the community boards at the [Lightning Platform Developer Boards](#) that describe how to implement compression with different SOAP stacks. The full Gzip analysis and discussion is available at Simon Fell's blog: <http://www.pocketsoap.com/weblog/2005/12/1583.html>.
- **Do Not Design a Multi-Threaded Client Application.** Multi-threading is not allowed for a single client application using the SOAP-based API.

Salesforce Settings

Most processing takes place in the database. Setting these parameters correctly will help the database process as quickly as possible:

- **Enable or Disable the Most Recently Used (MRU) functionality.** Records marked as most recently used (MRU) are listed in the “Recent Items” section of the sidebar in the Salesforce user interface. Check that you are not enabling it for calls where it is not needed.

In API version 7.0 and above, MRU functionality is disabled by default. To enable the MRU functionality, create this header and set the `updateMru` to `true`. The following sample shows how to use MRU functionality:

```
public void mruHeaderSample() {
    connection.setMruHeader(true);
    Account account = new Account();
    account.setName("This will be in the MRU");
    try {
        SaveResult[] sr = connection.create(new SObject[]{account});
        System.out.println("ID of account added to MRU: " +
            sr[0].getId());
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

- **Log in as a user with the “Modify All Data” permission to avoid sharing rules.** If the client application logs in as a user who has access to data via a sharing rule, then the API must issue an extra query to check access. To avoid this, log in as a user with the

“Modify All Data” permission. In general, fewer sharing rules quickens load speeds, as there are fewer operations that have to be performed when setting properties such as ownership.

Alternatively, you can set organization-wide defaults for some objects as public read/write for the duration of the load. For more information, see “Set Your Internal Organization-Wide Sharing Defaults” in the Salesforce online help.

- **Avoid workflow or assignment rules.** Anything that causes a post-operation action slows down the load. You can temporarily disable automatic rules if the loaded objects are exempt from them.
- **Avoid triggering cascading updates.** For example, if you update the owner of an account, the contacts and opportunities associated with that account may also require updates. Instead of updating a single object, the client application must access multiple objects, which slows down the load.

The Lightning Platform Data Loader is a good reference for data loading. It disables the MRU, uses HTTP/1.1 persistent connections, and applies GZIP compression on the request and response. If you are performing a data load, or are looking at a place to start when writing your own Java integration, the Lightning Platform Data Loader can serve as a fast and reliable solution. For more information about the Lightning Platform Data Loader, see: Data Loader in the Salesforce online help.

Best Practices with Any Data Loader

While this section presents a best practice process using the Lightning Platform Data Loader, the general principles apply to any client data loader:

1. Identify which data you will migrate.

You may not want or need to migrate a whole set of data—choose which objects you wish to migrate. For example, you may want to migrate only the contact information from each account, or only migrate account information from a particular division.

2. Create templates for the data.

Create one template for each object, for example in an Excel worksheet.

Identify the required fields for each object. In addition to the required fields for each standard object, there may be additional required fields such as those needed to follow business rules, or legacy ID fields. Use this guide or see the page layout definitions in the Salesforce user interface to find out which fields are required on standard objects.

You may wish to highlight the required fields in red for easier review of the data after you populate the templates.

You should also identify any ordering dependencies. Objects may have mandatory relationships, for example all accounts have an owner, and all opportunities are associated with an account. The dependencies in these relationships dictate the order of data migration. For Salesforce data, for example, you should load users first, then accounts, then opportunities.

To identify dependencies, review the related lists and lookup fields in the page layout of the given object, and IDs (foreign keys) in the database.

3. Populate the templates.

Clean your data before populating the template, and review the data in the templates.

4. Migrate the data.

Create custom fields to store legacy ID information. Optionally, give the custom field the `EXTERNAL ID` attribute so it will be indexed. This will help maintain relationships, and help you build custom reports for validation.

Load one record, check the results, then load all records.

5. Validate the data.

Use all of these techniques to validate your migration:

- Create custom reports that validate record counts and provide an overall snapshot of migration.
 - Spot check the data.
 - Review exception reports to see what data was not migrated.
6. Re-migrate or update data as needed.

Integration and Single Sign-On



Warning: To avoid getting into an unrecoverable state, do not enable single sign-on for your system administrator account. If you do, and then perform a single sign-on integration that fails, you may not be able to log in again to recover.

CHAPTER 17 Data Replication

In this chapter ...

- [API Calls for Data Replication](#)
- [Scope of Data Replication](#)
- [Data Replication Steps](#)
- [Object-Specific Requirements for Data Replication](#)
- [Polling for Changes](#)
- [Checking for Structural Changes in the Object](#)

The API supports data replication, which allows you to store and maintain an external, separate copy of your organization's pertinent Salesforce data for specialized uses, such as data warehousing, data mining, custom reporting, analytics, and integration with other applications. Data replication provides you with local control and the ability to run large or ad hoc analytical queries across the entire data set without transmitting all that data across the network.



Note: To get real-time notifications of Salesforce record changes, use Change Data Capture instead. By subscribing to a Change Data Capture channel, you receive a stream of change event messages for record changes, including insertions, updates, deletions, and undeletions. With Change Data Capture, you get broad access to data and can perform updates in your target store using transaction boundaries. Change Data Capture provides a versioned event schema and retains change events temporarily for later retrieval. For more information, see the [Change Data Capture Basics Trailhead module](#), or for a complete reference, see the [Change Data Capture Developer Guide](#).

Use the topics in this section to better understand the best practices for data replication.

API Calls for Data Replication

The API supports data replication with the following API calls:

API Call	Description
<code>getUpdated()</code>	Retrieves the list of objects that have been updated (added or changed) during the specified timespan for the specified object.
<code>getDeleted()</code>	Retrieves the list of objects that have been deleted during the specified timespan for the specified object.

Client applications can invoke these API calls to determine which objects in your organization's data have been updated or deleted during a given time period. These API calls return a set of IDs for objects that have been updated (added or changed) or deleted, as well as the timestamp (Coordinated Universal Time (UTC)—not local—timezone) indicating when they were last updated or deleted. It is the responsibility of the client application to process these results and to incorporate the required changes into the local copy of the data.

Scope of Data Replication

This feature provides a mechanism that targets data replication (one-way copying of data). It does not provide data synchronization (two-way copying of data) or data mirroring capabilities.

Data Replication Steps

The following is a typical data replication procedure for an object:

1. Optionally, determine whether the structure of the object has changed since the last replication request, as described in [Checking for Structural Changes in the Object](#).
2. Call `getUpdated()`, passing in the object and timespan for which to retrieve data.
Note that `getUpdated()` retrieves the IDs for data to which the logged in user has access. Data that is outside of the user's sharing model is not returned. The API returns the ID of every changed object that is visible to you, regardless of what change occurred in the object. For information on IDs, see [ID Field Type](#).
3. Pass in all IDs in an array. For each ID element in the array, call `retrieve()` to obtain the latest information you want from the associated object. You must then take the appropriate action on the local data, such as inserting new rows or updating existing ones with the latest information.
4. Call `getDeleted()`, passing in the object and timespan for which to retrieve data. Like `getUpdated()`, `getDeleted()` retrieves the IDs for data to which the logged-in user has access. Data that is outside of the user's sharing model is not returned. The API returns the ID of every changed object that is visible to you, regardless of what change occurred in the object, based on `SystemModstamp` field information if available. For information on IDs, see [ID Field Type](#).
5. Iterate through the returned array of IDs. Your client application must then take the appropriate action on the local data to remove (or flag as deleted) the deleted objects. If your client application cannot match rows in the local data using the retrieved object ID, then the local data rows either were deleted or were never created, in which case there is nothing to do.
6. Optionally, save the request time spans for future reference. You can do this with the `getDeleted()` `latestDateCovered` value or the `getUpdated()` `latestDateCovered` value.

Object-Specific Requirements for Data Replication

The API objects have the following requirements for data replication:

- The `getUpdated()` and `getDeleted()` calls filter the results so that the client application receives IDs for only those created or updated objects to which the logged-in user has access. For information on IDs, see [ID Field Type](#).
- Your client application can replicate any objects to which it has sufficient permissions. For example, to replicate all data for your organization, your client application must be logged in with the “View All Data” permission. For more information, see [Factors that Affect Data Access](#).
- The logged-in user must have read access to the object. For more information, see “Set Your Internal Organization-Wide Sharing Defaults” in Salesforce Help.
- The object must be configured to be replicatable (`replicatable` is `true`). To determine whether a given object can be replicated, your application can invoke the `describeSObject()` call on the object and inspect the `replicatable` property in the `describeSObjectResult`.

Polling for Changes

Client applications typically poll for changed data periodically. Polling involves the following considerations:

- The polling frequency depends on business requirements for how quickly changes in your organization’s Salesforce data need to be reflected in the local copy. Some client applications might poll once a day to retrieve changes, while other client applications might poll every five minutes to achieve closer accuracy.
- Deleted records are written to a delete log, which `getDeleted()` accesses. A background process that runs every two hours purges records that have been in an organization’s delete log for more than two hours if the number of records is above a certain limit. Starting with the oldest records, the process purges delete log entries until the delete log is back below the limit. This is done to protect Salesforce from performance issues related to massive delete logs. The limit is calculated using this formula:

```
5000 * number of licenses in the organization
```

For example, an organization with 1,000 licenses could have up to 5,000,000 (five million) records in the delete log before any purging took place. If purging has been performed before your `getDeleted()` call is executed, an `INVALID_REPLICATION_DATE` error is returned. If you get this exception, you should do a full pull of the table.

- The API truncates the seconds portion of `dateTime` values. For example, if a client application submits a timespan between 12:30:15 and 12:35:15 (Coordinated Universal Time (UTC) time), then the API retrieves information about items that have changed between 12:30:00 and 12:35:00 (UTC), inclusive.
 - 📌 **Note:** Development tools differ in the way that they handle time data. Some development tools report the local time, while others report only the Coordinated Universal Time (UTC) time. To determine how your development tool handles time values, refer to its documentation.
- We recommend polling no more frequently than every five minutes. There are built in controls to prevent errant applications from invoking the data replication API calls too frequently.
- Client applications should save the timespan used in previous data replication API calls so that the application knows the last time period for which data replication was successfully completed.
- To ensure data integrity on the local copy of the data, a client application needs to capture all of the relevant changes during polling—even if it requires processing data redundantly to ensure that there are no gaps. Your client application can contain business logic to skip processing objects that have already been integrated into your local data.

- Gaps can also occur if the client application somehow fails to poll the data as expected (for example, due to a hardware crash or network connection failure). Your client application can contain business logic that determines the last successful replication and polls for the next consecutive timespan.
- If for any reason the local data is compromised, your client application might also provide business logic for rebuilding the local data from scratch.

 **Note:** You can now use [Outbound Messaging](#) to trigger actions instead of polling for them.

Checking for Structural Changes in the Object

In the API, data replication only reflects changes made to object records. It does not determine whether changes have been made to the structure of objects (for example, fields added to—or removed from—a custom object). It is the responsibility of the client application to check whether the structure of a given object has changed since the last update. Before replicating data, client applications can call `describeObjects()` on the object, and then compare the data returned in the `DescribeObjectResult` with the data returned and saved from previous `describeObjects()` invocations.

CHAPTER 18 Feature-Specific Considerations

In this chapter ...

- [Archived Activities](#)
- [Person Account Record Types](#)
- [External Objects](#)
- [Call Centers and the API](#)
- [Implementing Salesforce Integrations on Lightning Platform](#)
- [Knowledge](#)

Some Salesforce features require special consideration when accessed via the API. Use the topics in this section to learn about the special considerations for activities, person accounts, forecast override business rules, the Call Center, and creating your own apps.

Archived Activities

Salesforce archives activities (tasks and events) that are over a year old.

You can use `queryAll()` to query on all `Task` and `Event` records, archived or not. You can also filter on the `isArchived` field to find only the archived objects. You cannot use `query()` as it automatically filters out all records where `isArchived` is set to `true`. You can update or delete archived records, though you cannot update the `isArchived` field. If you use the API to insert activities that meet the criteria listed below, the activities will be archived during the next run of the archival background process.

Older `Events` and `Tasks` are archived according to the criteria listed below. In the Salesforce user interface, users can view archived activities in several locations.

- Click **View All** in the Activity History related list to open the Activity History tab. In the Activity History tab, you can sort entries and open, edit, or delete activities.
- Click **View All** in the activity timeline to open the All Activity History list. Up to 2,000 records appear, including archived records. The All Activity History list is ideal for printing.

In the API, archived activities can only be queried via `queryAll()`.

Activity archive criteria:

- Events with an `ActivityDateTime` or `ActivityDate` value greater than or equal to 365 days old
- Tasks with an `IsClosed` value of `true` and an `ActivityDate` value greater than or equal to 365 days old
- `Tasks` with an `IsClosed` value of `true`, a blank `ActivityDate` field, and a create date greater than or equal to 365 days ago

For more information, see [View Archived Activities in Salesforce Help](#).

Person Account Record Types

Beginning with API version 8.0, a new family of record types on `Account` objects is available: “person account” record types. The person account record types enable specialized business-to-consumer functionality for users who sell to or do business with individuals. For example, a doctor, hairdresser, or real estate agent whose clients are individuals. For more information about person accounts, see “Person Accounts” and “Considerations for Using Person Accounts” in the Salesforce Help.

Record types are person account record types if the `Account` field `IsPersonAccount` is set to `true`. Salesforce provides one default person account record type, `PersonAccount`, but an administrator can create additional person account record types. Conversely, record types with the `Account` field `IsPersonAccount` set to `false` are “business account” record types, which are traditional business-to-business (B2B) Salesforce accounts.

When a person account is created (or an existing business account is changed to a person account), a corresponding contact record is also created. This contact record is referred to as a “person contact.” The person contact enables the person account to function simultaneously as both an account and a contact. This record is the only contact record that can be associated directly with the person account. Also, the ID of the corresponding person contact record is stored in the `PersonContactId` field on the person account.

Review this list of facts about person account record types before working with them.

- Contact your account representative to enable the person account feature, if the feature isn’t enable yet.
- You can use a query similar to the following example to find all records with a person account record type:

```
SELECT Name, SubjectType, IsPersonType
FROM RecordType
WHERE SubjectType='Account' AND IsPersonType=True
```

- If you issue a `query()` call against an account, the results return the root object type in the `ObjectType` field. The returned value is always `Account`.
- A person contact can be modified, but cannot be created or deleted. Since these kinds of contacts do not have their own record detail page, clients must redirect users to the corresponding person account (`Account`) page. SOSL results don't include any of the contact fields enabled when `IsPersonAccount` is set to `true`. The contact `ReportsToId` field is not visible.
- If you delete the account, the contact is also deleted. You cannot directly delete the contact; you must delete the account.
- You can change the record type of an account across record type families (typically performed when migrating business accounts to person accounts, but the reverse operation is also supported). When you change the record type from a business account to a person account, the person contact is created. When you change the record type from a person account to a business account, the person fields are set to null, and the person contact becomes a regular contact with the same parent account it had before the change.



Note: You cannot change record types across record type families in the Salesforce user interface.

- If you change the record type of a business account to a person account using either `update()` or `upsert()`, you cannot make any other changes to fields in that account in the same call; if attempted, the fault `INVALID_FIELD_FOR_INSERT_UPDATE` results. However, you can change record type values from one person account record type to another, or from one business account record type to another, in the same call with other changes.
- When converting a business account to a person account, there must be a one-to-one relationship between each business account record and its corresponding contact record. Furthermore, fields common to both records such as `Owner` and `Currency` must have identical values.
- Workflow and validation formulas do not fire during a change in record types from or to person accounts.
- When you change a business account to a person account, valid records are changed and invalid records show an error in the results array.
- When you change a person account to a business account, no validation is performed.
- `describeLayout()` for version 7.0 and below returns the default business account record type as the default record type even if the tab default is a person account record type. In version 8.0 and after, it will always be the tab default.
- `describeLayout()` for version 7.0 and below doesn't return any person account record types.
- `describeSObject()` for version 7.0 and below show `Account` objects as not creatable if the profile does not have access to any business record types.
- After conversion, the new person accounts will have unique one-to-one relationships with the contact records that formed them. As is true for all person accounts, no other contacts can be associated to a person account.
- After conversion, any existing account field history information remains on the person accounts. Any existing contact field history information is retained on the contact, but is not added to the person accounts field history.

For more information about person accounts, see the Salesforce Help.

External Objects

Special behaviors and limitations apply to `queryAll()` and `queryMore()` calls on external data.

`queryAll()`

Because Salesforce doesn't track changes to external data, `queryAll()` behaves the same as `query()` for external objects.

queryMore()

It's common for Salesforce Connect queries of external data to have a large result set that's broken into smaller batches or pages. When querying external objects, Salesforce Connect accesses the external data in real time via Web service callouts. Each `queryMore()` call results in a Web service callout. The batch boundaries and page sizes depend on your adapter and how you set up the external data source.

We recommend the following:

- When possible, avoid paging by filtering your queries of external objects to return fewer rows than the batch size, which by default is 500 rows. Remember, obtaining each batch requires a `queryMore()` call, which results in a Web service callout.
- If the external data frequently changes, avoid using `queryMore()` calls. If the external data is modified between `queryMore()` calls, you can get an unexpected `QueryResult`.

If the primary or “driving” object for a `SELECT` statement is an external object, `queryMore()` supports only that primary object and doesn't support subqueries.

By default, the OData 2.0 and 4.0 adapters for Salesforce Connect use client-driven paging. With client-driven paging, OData adapters convert each `queryMore()` call into an OData query that uses the `$skip` and `$top` system query options to specify the batch boundary and page size. These options are similar to using `LIMIT` and `OFFSET` clauses to page through a result set.

If you enable server-driven paging on an external data source, Salesforce ignores the requested page sizes, including the default `queryMore()` batch size of 500 rows. The pages returned by the external system determine the batches, but each page can't exceed 2,000 rows.

Call Centers and the API

The API provides access to information about computer–telephony integration (CTI) call centers with the `describeSoftphoneLayout()` call. You must have the CTI feature enabled for your organization. Contact your account representative for assistance.

The API supports limited access to call center-related objects, including being able to create call centers, and create or modify additional numbers for the call center.

Topic	Description
CallCenter	Call Center object description, including fields and usage.
AdditionalNumber	Configuration settings that allow you to add an additional number if it cannot easily be categorized as a user, contact, lead, account, or any other object. Examples include phone queues or conference rooms.

In addition, several fields have been added to existing objects to support call centers. The following fields provide configuration settings for operation of a call center.

Object Name	Field Name	Field Type	Field Properties	Description
OpenActivity ActivityHistory Task	CallDisposition	string	Create (Task only) Filter	Represents the result of a given call, for example, “we'll call back,” or “call unsuccessful.” Limit is 255 characters.

Object Name	Field Name	Field Type	Field Properties	Description
			Nillable Update (Task only)	For the Task object, corresponds to the Salesforce user interface label Call Result . You can also create and update values for this field in Task.
OpenActivity ActivityHistory Task	CallDurationInSeconds	int	Create (Task only) Filter Nillable Update (Task only)	Duration of the call in seconds. For Task, you can also create and update values for this field.
OpenActivity ActivityHistory Task	CallObject	string	Filter Nillable Update (Task only)	Name of a call center. Limit is 255 characters. For Task, you can also create and update values for this field.
OpenActivity ActivityHistory Task	CallType	picklist	Create (Task only) Filter Nillable Restricted picklist Update	The type of call being answered: Inbound, Internal, or Outbound. For Task, you can also create and update values for this field.
User	CallCenterId	reference	Create Filter Nillable Update	The unique identifier for the call center associated with this user.
User	UserPermissionsCallCenterAutoLogin	boolean	Create Update	Indicates whether a user will be automatically logged in to a call center when logging in to the Salesforce application (<code>true</code>) or not (<code>false</code>).

Implementing Salesforce Integrations on Lightning Platform

You can implement your Salesforce integrations or other client applications, on the Lightning Platform by creating a Salesforce AppExchange app.

1. Create a [WebLink](#) that passes the user session ID and the API server URL to an external site:

```
https://www.your_tool.com/test.jsp?sessionId={!API_Session_ID}&url={!API_Partner_Server_URL_80}
```

Use `https` to ensure that your session ID cannot be detected.

2. The page pointed to in the preceding step takes the session ID and uses it to call back to the API. Use `getUserInfo()` to return the `userID` associated with the session and related information. If needed, you can also use `retrieve` on the `User` object to retrieve any additional information you need about the user.
3. Maintain a cross-reference between the `UserID` or `username` and the corresponding user ID in your system, which you can do using a [WebLink](#) that is executed when the user clicks a tab, or a [WebLink](#) on the page layout.
4. Package and upload this app using the instructions in Salesforce Help topic “Prepare Your Apps for Distribution.”

Accessing Salesforce Data Using the API and OAuth

Salesforce supports OAuth 1.0.A and 2.0 for SOAP API requests.

Using an already defined connected app and the OAuth protocol, a third party can implement an OAuth authentication flow to integrate with the Salesforce API.

For detailed steps about integrating with the Salesforce API using OAuth, see [Authenticating Apps with OAuth](#) in Salesforce Help.

Partners, who wish to get an OAuth consumer Id for authentication, can contact Salesforce

Knowledge

Articles Overview

Articles capture information about your company's products and services that you want to make available in your knowledge base. Articles in the knowledge base can be classified using data categories to make it easy for users to find the articles they need. Administrators can use data categories to control access to articles.

Knowledge Articles vs. Knowledge Article Versions

When working with articles, keep in mind that the [KnowledgeArticle](#) represents the parent record of all article versions. [KnowledgeArticleVersion](#) records represent each version of a given article.

Record Types vs. Article Types

The article structure is represented differently between Lightning Experience and Salesforce Classic. In Lightning Knowledge, you use the same record types available in other custom objects (see the `RecordTypeId` field on [Knowledge__kav](#)) to structure different types of articles. For example, you can use different layouts for different record types. In Salesforce Classic, you get this functionality through article types (see the `ArticleType` field on [KnowledgeArticleVersion](#)). Each unique type of article has a unique object in Salesforce Classic (for example, `FAQ__kav` for FAQ article types). Lightning Knowledge does not have a unique object for each type because it is handled using the record type.

Audience Channel

An audience, sometimes called a channel, refers to the types of users who can access an article. Salesforce Knowledge offers four channels where you can make articles available.

- **Internal App:** Salesforce users can access articles depending on their role visibility.
- **Customer:** Customers can access articles in a community, site, or customer portal. Customer users inherit the role visibility of the manager on the account. In a community, the article is only available to users with Customer Community or Customer Community Plus licenses.
- **Partner:** Partners can access articles in a community, site, or partner portal. Partner users inherit the role visibility of the manager on the account. In a community, the article is only available to users with Partner Community licenses.
- **Public Knowledge Base:** Articles can be made available to anonymous users by creating a public knowledge base. With Lightning Knowledge, most Salesforce orgs use Communities to create a knowledge base. Creating a public knowledge base for Salesforce Knowledge in Salesforce Classic requires Sites and Visualforce.

Publishing Cycle

Salesforce Knowledge Articles move through a publishing cycle from their creation to their deletion. The publishing cycle includes three different statuses: `DRAFT` is the stage when a new article is being created or an existing one is being updated. Articles with the `Online` status are draft articles that have been published and are now available to their different channels. Eventually, when a published article is at the end of its life, it can be moved to the `Archived` status or sent back to `DRAFT` to be updated in a subsequent version.

Working with Articles in the API

Articles are available through the [KnowledgeArticleVersion](#) and [KnowledgeArticle](#) objects in the API. They both represent an article but provide different capabilities.

KnowledgeArticleVersion

Every new draft article in Salesforce Knowledge has a version number. When an article is published and you want to update it, you can create a new `DRAFT` with a distinct version number. Each version has its own ID. Once the updated version is ready to be published, it replaces the former one and updates the version number. You can access the content of an article version using the `KnowledgeArticleVersion` object and filter on its `Draft` or `Online` status. For example, the following query returns the title of the `Draft` version of all the articles across all article types in United States English:

```
SELECT Title
FROM KnowledgeArticleVersion
WHERE PublishStatus='Draft'
AND language = 'en_US'
```

Articles are also auto-assigned an Article Number, which is not a unique identifier to an individual article, but an identifier to a master article and all of its available translations.

 **Note:** Both the master version (the knowledge article with `IsMasterLanguage = 1`) and the translations are `KnowledgeArticleVersion` objects.

KnowledgeArticle

Unlike `KnowledgeArticleVersion`, the ID of a `KnowledgeArticle` record is identical irrespective of the article's version (status). Where the `KnowledgeArticleVersion` object provides API access to an article's custom field values, the `KnowledgeArticle` object provides API access to an article's metadata fields.

The article record is the parent container of all versions of an article, whatever the publishing status (draft, published, archived) and the language. While `KnowledgeArticle` and `KnowledgeArticleVersion` represent any article in the knowledge base, use the concrete representation of these objects for the specific articles. In Lightning Knowledge, these concrete representations default to `Knowledge__ka`

(for the Knowledge article) and Knowledge__kav (for the Knowledge article version). In Salesforce Classic, use <Article Type>__ka and <Article Type>__kav.

The following Lightning Knowledge query returns the title for all the published FAQ articles in United States English:

```
SELECT Title
FROM Knowledge__kav
WHERE PublishStatus='online'
AND Language = 'en_US'
AND RecordTypeId = '<specify RecordTypeId for FAQ here>'
```

The following Salesforce Classic query returns the title for all the published offers in United States English:

```
SELECT Title
FROM FAQ__kav
WHERE PublishStatus='online'
AND language ='en_US'
```

Data Categories Overview

Data categories are organized by category group and let:

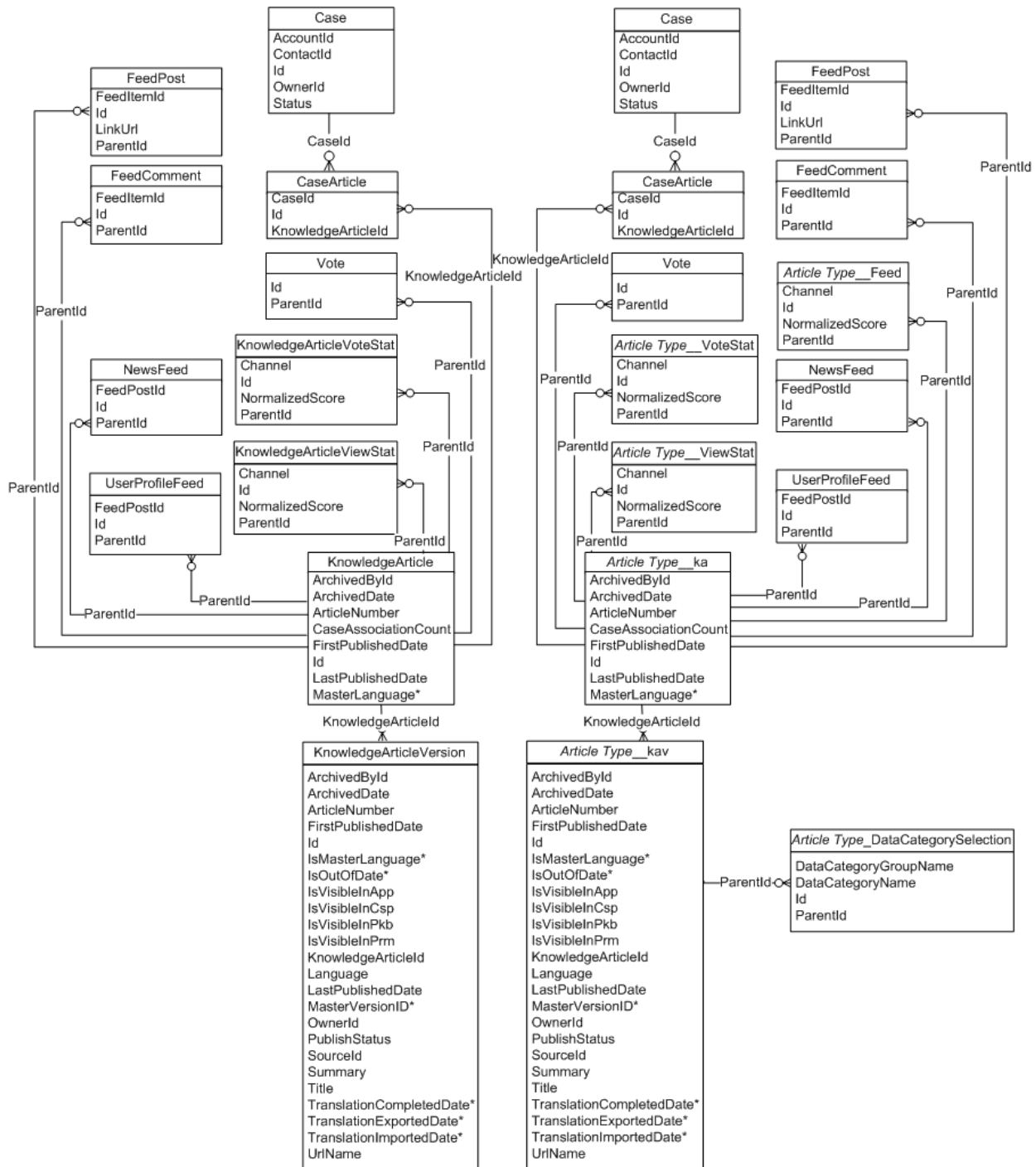
- Users classify and find records.
- Administrators control access to records.

Salesforce Knowledge uses data categories to classify articles and make them easier to find. For example, to classify articles by sales regions and products, create two category groups: Sales Regions, Products. The Sales Regions category group could consist of a geographical hierarchy, such as All Sales Regions as the top level and North America, Europe, and Asia at the second level. The Products group could have All Products as the top level and Phones, Computers, and Printers at the second.

Working with Data Categories in the API

The following table lists API resources for working with data categories.

Name	Type	Description
Knowledge__DataCategorySelection	Object	Gives access to article categorization in Lightning Knowledge.
Article Type__DataCategorySelection	Object	Gives access to article categorization in Knowledge for Salesforce Classic.
QuestionDataCategorySelection	Object	Gives access to question categorization.
WITH DATA CATEGORY filteringExpression	SOQL clause	Filters articles depending on their status in the publishing cycle and their data categories. For more information, see the Salesforce SOQL and SOSL Reference Guide .
WITH DATA CATEGORY DataCategorySpec	SOSL clause	Finds articles based on their categorization. For more information, see the Salesforce SOQL and SOSL Reference Guide .
describeDataCategoryGroups ()	Call	Retrieves available category groups for objects specified in the request.
describeDataCategoryGroupStructures ()	Call	Retrieves available category groups along with their data category structure for objects specified in the request.



Additional Information

To learn more about managing your knowledge base using the API, see the [Knowledge Developer Guide](#).

GLOSSARY

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#) | [Y](#) | [Z](#)

A

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](#).

Anonymous Block, Apex

Apex code that does not get stored in Salesforce, but that can be compiled and executed by using the `ExecuteAnonymousResult()` API call, or the equivalent in the AJAX Toolkit.

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.

Apex

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

Apex-Managed Sharing

Enables developers to programmatically manipulate sharing to support their application's behavior. Apex-managed sharing is only available for custom objects.

App

Short for "application." A collection of components such as tabs, reports, dashboards, and Visualforce pages that address a specific business need. Salesforce provides standard apps such as Sales and Service. You can customize the standard apps to match the way you work. In addition, you can package an app and upload it to the AppExchange along with related components such as custom fields, custom tabs, and custom objects. Then, you can make the app available to other Salesforce users from the AppExchange.

AppExchange

The AppExchange is a sharing interface from Salesforce that allows you to browse and share apps and services for the Lightning Platform.

AppExchange Upgrades

Upgrading an app is the process of installing a newer version.

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.

B

Boolean Operators

You can use Boolean operators in report filters to specify the logical relationship between two values. For example, the AND operator between two values yields search results that include both values. Likewise, the OR operator between two values yields search results that include either value.

Bulk API 2.0

The REST-based Bulk API 2.0 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 job that is processed in the background by Salesforce. See also SOAP API.

C

Callout, Apex

An Apex callout enables you to tightly integrate your Apex with an external service by making a call to an external Web service or sending a HTTP request from Apex code and then receiving the response.

Child Relationship

A relationship that has been defined on an sObject that references another sObject as the “one” side of a one-to-many relationship. For example, contacts, opportunities, and tasks have child relationships with accounts.

See also sObject.

Class, Apex

A template or blueprint from which Apex objects are created. Classes consist of other classes, user-defined methods, variables, exception types, and static initialization code. In most cases, Apex classes are modeled on their counterparts in Java.

Client App

An app that runs outside the Salesforce user interface and uses only the Lightning Platform API or Bulk API 2.0. 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.

Component, Visualforce

Something that can be added to a Visualforce page with a set of tags, for example, `<apex:detail>`. Visualforce includes a number of standard components, or you can create your own custom components.

Component Reference, Visualforce

A description of the standard and custom Visualforce components that are available in your organization. You can access the component library from the development footer of any Visualforce page or the [Visualforce Developer's Guide](#).

Controller, Visualforce

An Apex class that provides a Visualforce page with the data and business logic it needs to run. Visualforce pages can use the standard controllers that come by default with every standard or custom object, or they can use custom controllers.

Controlling Field

Any standard or custom picklist or checkbox field whose values control the available values in one or more corresponding dependent fields.

Custom App

See App.

Custom Field

A field that can be added in addition to the standard fields to customize Salesforce for your organization's needs.

Custom Help

Custom text administrators create to provide users with on-screen information specific to a standard field, custom field, or custom object.

Custom Links

Custom links are URLs defined by administrators to integrate your Salesforce data with external websites and back-office systems. Formerly known as Web links.

Custom Object

Custom records that allow you to store information unique to your organization.

Custom S-Control

Note: S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

Custom Web content for use in custom links. Custom s-controls can contain any type of content that you can display in a browser, for example a Java applet, an Active-X control, an Excel file, or a custom HTML Web form.

D

Database

An organized collection of information. The underlying architecture of the Lightning Platform includes a database where your data is stored.

Database Table

A list of information, presented with rows and columns, about the person, thing, or concept you want to track. See also Object.

Data Loader

A Lightning Platform tool used to import and export data from your Salesforce organization.

Data Manipulation Language (DML)

An Apex method or operation that inserts, updates, or deletes records.

Date Literal

A keyword in a SOQL or SOSL query that represents a relative range of time such as `last month` or `next year`.

Decimal Places

Parameter for number, currency, and percent custom fields that indicates the total number of digits you can enter to the right of a decimal point, for example, 4.98 for an entry of 2. Note that the system rounds the decimal numbers you enter, if necessary. For example, if you enter 4.986 in a field with `Decimal Places` of 2, the number rounds to 4.99. Salesforce uses the round half-up rounding algorithm. Half-way values are always rounded up. For example, 1.45 is rounded to 1.5. -1.45 is rounded to -1.5.

Delegated Authentication

A security process where an external authority is used to authenticate Lightning Platform users.

Dependent Field

Any custom picklist or multi-select picklist field that displays available values based on the value selected in its corresponding controlling field.

Developer Edition

A free, fully-functional Salesforce organization designed for developers to extend, integrate, and develop with the Lightning Platform. Developer Edition accounts are available on developer.salesforce.com.

Salesforce Developers

The Salesforce Developers website at developer.salesforce.com provides a full range of resources for platform developers, including sample code, toolkits, an online developer community, and the ability to obtain limited Lightning Platform environments.

Document Library

A place to store documents without attaching them to accounts, contacts, opportunities, or other records.


E

Email Alert

Email alerts are actions that send emails, using a specified email template, to specified recipients.

Email Template

A form email that communicates a standard message, such as a welcome letter to new employees or an acknowledgment that a customer service request has been received. Email templates can be personalized with merge fields, and can be written in text, HTML, or custom format.

 **Note:** Lightning email templates aren't packageable.

Enterprise Edition

A Salesforce edition designed for larger, more complex businesses.

Enterprise WSDL

A strongly-typed WSDL for customers who want to build an integration with their Salesforce organization only, or for partners who are using tools like Tibco or webMethods to build integrations that require strong typecasting. The downside of the Enterprise WSDL is that it only works with the schema of a single Salesforce organization because it is bound to all of the unique objects and fields that exist in that organization's data model.

Entity Relationship Diagram (ERD)

A data modeling tool that helps you organize your data into entities (or objects, as they are called in the Lightning Platform) and define the relationships between them. [ERDs](#) for key Salesforce objects are published in the *Salesforce Object Reference*.

F

Field

A part of an object that holds a specific piece of information, such as a text or currency value.

Field-Level Security

Settings that determine whether fields are hidden, visible, read only, or editable for users. Available in Professional, Enterprise, Unlimited, Performance, and Developer Editions.

Filter Condition/Criteria

Condition on particular fields that qualifies items to be included in a list view or report, such as "State equals California."

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.

Formula Field

A type of custom field. Formula fields automatically calculate their values based on the values of merge fields, expressions, or other values.

Function

Built-in formulas that you can customize with input parameters. For example, the DATE function creates a date field type from a given year, month, and day.

G

Gregorian Year

A calendar based on a 12-month structure used throughout much of the world.

Group Edition

A product designed for small businesses and workgroups with a limited number of users.

H

HTTP Debugger

An application that can be used to identify and inspect SOAP requests that are sent from the AJAX Toolkit. They behave as proxy servers running on your local machine and allow you to inspect and author individual requests.

I

ID

See Salesforce Record ID.

Inline S-Control

Note: S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

An s-control that displays within a record detail page or dashboard, rather than on its own page.

Instance

The cluster of software and hardware represented as a single logical server that hosts an organization's data and runs their applications. The Lightning Platform runs on multiple instances, but data for any single organization is always stored on a single instance.

Integration User

A Salesforce user defined solely for client apps or integrations. Also referred to as the logged-in user in a SOAP API context.

ISO Code

The International Organization for Standardization country code, which represents each country by two letters.

J

Junction Object

A custom object with two master-detail relationships. Using a custom junction object, you can model a "many-to-many" relationship between two objects. For example, you create 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.

K

No Glossary items for this entry.

L

License Management Application (LMA)

A free AppExchange app that allows you to track sales leads and accounts for every user who downloads your managed package (app) from the AppExchange.

License Management Organization (LMO)

The Salesforce organization that you use to track all the Salesforce users who install your package. A license management organization must have the License Management Application (LMA) installed. It automatically receives notification every time your package is installed or uninstalled so that you can easily notify users of upgrades. You can specify any Enterprise, Unlimited, Performance, or Developer Edition organization as your license management organization. For more information, go to [Managing Licenses for Managed Packages](#).

Lightning Platform

The Salesforce platform for building applications in the cloud. Lightning Platform combines a powerful user interface, operating system, and database to allow you to customize and deploy applications in the cloud for your entire enterprise.

List View

A list display of items (for example, accounts or contacts) based on specific criteria. Salesforce provides some predefined views.

In the Agent console, the list view is the top frame that displays a list view of records based on specific criteria. The list views you can select to display in the console are the same list views defined on the tabs of other objects. You cannot create a list view within the console.

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`.

Logged-in User

In a SOAP API context, the username used to log into Salesforce. Client applications run with the permissions and sharing of the logged-in user. Also referred to as an integration user.

M

Managed Package

A collection of application components that is posted as a unit on the AppExchange and associated with a namespace and possibly a License Management Organization. To support upgrades, a package must be managed. An organization can create a single managed package that can be downloaded and installed by many different organizations. Managed packages differ from unmanaged packages by having some locked components, allowing the managed package to be upgraded later. Unmanaged packages do not include locked components and cannot be upgraded. In addition, managed packages obfuscate certain components (like Apex) on subscribing organizations to protect the intellectual property of the developer.

Manual Sharing

Record-level access rules that allow record owners to give read and edit permissions to other users who might not have access to the record any other way.

Many-to-Many Relationship

A relationship where each side of the relationship can have many children on the other side. Many-to-many relationships are implemented through the use of junction objects.

Master-Detail Relationship

A relationship between two different types of records that associates the records with each other. For example, accounts have a master-detail relationship with opportunities. This type of relationship affects record deletion, security, and makes the lookup relationship field required on the page layout.

Master Picklist

A complete list of picklist values available for a record type or business process.

Metadata

Information about the structure, appearance, and functionality of an organization and any of its parts. Lightning Platform uses XML to describe metadata.

Metadata WSDL

A WSDL for users who want to use the Lightning Platform Metadata API calls.

Multitenancy

An application model where all users and apps share a single, common infrastructure and code base.

N

Namespace

In a packaging context, a one- to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other developers on AppExchange, similar to a domain name. Salesforce automatically prepends your namespace prefix, followed by two underscores ("__"), to all unique component names in your Salesforce organization.

Native App

An app that is built exclusively with setup (metadata) configuration on Lightning Platform. Native apps do not require any external services or infrastructure.

O

Object

An object allows you to store information in your Salesforce organization. The object is the overall definition of the type of information you are storing. For example, the case object allow you to store information regarding customer inquiries. For each object, your organization will have multiple records that store the information about specific instances of that type of data. For example, you might have a case record to store the information about Joe Smith's training inquiry and another case record to store the information about Mary Johnson's configuration issue.

Object-Level Help

Custom help text that you can provide for any custom object. It displays on custom object record home (overview), detail, and edit pages, as well as list views and related lists.

Object-Level Security

Settings that allow an administrator to hide whole objects from users so that they don't know that type of data exists. Object-level security is specified with object permissions.

onClick JavaScript

JavaScript code that executes when a button or link is clicked.

One-to-Many Relationship

A relationship in which a single object is related to many other objects. For example, an account may have one or more related contacts.

Organization-Wide Defaults

Settings that allow you to specify the baseline level of data access that a user has in your organization. For example, you can set organization-wide defaults so that any user can see any record of a particular object that is enabled via their object permissions, but they need extra permissions to edit one.

Outbound Call

Any call that originates from a user to a number outside of a call center in Salesforce CRM Call Center.

Outbound Message

An outbound message sends information to a designated endpoint, like an external service. Outbound messages are configured from Setup. You must configure the external endpoint and create a listener for the messages using the SOAP API.

Overlay

An overlay displays additional information when you hover your mouse over certain user interface elements. Depending on the overlay, it will close when you move your mouse away, click outside of the overlay, or click a close button.

Owner

Individual user to which a record (for example, a contact or case) is assigned.

P

PaaS

See Platform as a Service.

Package

A group of Lightning Platform components and applications that are made available to other organizations through the AppExchange. You use packages to bundle an app along with any related components so that you can upload them to AppExchange together.

Package Dependency

This is created when one component references another component, permission, or preference that is required for the component to be valid. Components can include but are not limited to:

- Standard or custom fields
- Standard or custom objects
- Visualforce pages
- Apex code

Permissions and preferences can include but are not limited to:

- Divisions
- Multicurrency
- Record types

Package Installation

Installation incorporates the contents of a package into your Salesforce organization. A package on the AppExchange can include an app, a component, or a combination of the two. After you install a package, you may need to deploy components in the package to make it generally available to the users in your organization.

Package Publication

Publishing your package makes it publicly available on the AppExchange.

Package Version

A package version is a number that identifies the set of components uploaded in a package. The version number has the format *majorNumber.minorNumber.patchNumber* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The *patchNumber* is generated and updated only for a patch release.

Unmanaged packages are not upgradeable, so each package version is simply a set of components for distribution. A package version has more significance for managed packages. Packages can exhibit different behavior for different versions. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package. See also Patch and Patch Development Organization.

Parent Account

An organization or company that an account is affiliated. By specifying a parent for an account, you can get a global view of all parent/subsidiary relationships using the **View Hierarchy** link.

Partner WSDL

A loosely-typed WSDL for customers, partners, and ISVs who want to build an integration or an AppExchange app that can work across multiple Salesforce organizations. With this WSDL, the developer is responsible for marshaling data in the correct object representation, which typically involves editing the XML. However, the developer is also freed from being dependent on any particular data model or Salesforce organization. Contrast this with the Enterprise WSDL, which is strongly typed.

Patch

A patch enables a developer to change the functionality of existing components in a managed package, while ensuring subscribing organizations that there are no visible behavior changes to the package. For example, you can add new variables or change the body of an Apex class, but you may not add, deprecate, or remove any of its methods. Patches are tracked by a *patchNumber* appended to every package version. See also Patch Development Organization and Package Version.

Patch Development Organization

The organization where patch versions are developed, maintained, and uploaded. Patch development organizations are created automatically for a developer organization when they request to create a patch. See also Patch and Package Version.

Personal Edition

Product designed for individual sales representatives and single users.

Personal Information

User information including personal contact information, quotas, personal group information, and default opportunity team.

Picklist

Selection list of options available for specific fields in a Salesforce object, for example, the `Industry` field for accounts. Users can choose a single value from a list of options rather than make an entry directly in the field. See also Master Picklist.

Picklist (Multi-Select)

Selection list of options available for specific fields in a Salesforce object. Multi-select picklists allow users to choose one or more values. Users can choose a value by double clicking on it, or choose additional values from a scrolling list by holding down the CTRL key while clicking a value and using the arrow icon to move them to the selected box.

Picklist Values

Selections displayed in drop-down lists for particular fields. Some values come predefined, and other values can be changed or defined by an administrator.

Platform as a Service (PaaS)

An environment where developers use programming tools offered by a service provider to create applications and deploy them in a cloud. The application is hosted as a service and provided to customers via the Internet. The PaaS vendor provides an API for creating and extending specialized applications. The PaaS vendor also takes responsibility for the daily maintenance, operation, and support of the deployed application and each customer's data. The service alleviates the need for programmers to install, configure,

and maintain the applications on their own hardware, software, and related IT resources. Services can be delivered using the PaaS environment to any market segment.

Platform Edition

A Salesforce edition based on Enterprise, Unlimited, or Performance Edition that does not include any of the standard Salesforce apps, such as Sales or Service & Support.

Primary Key

A relational database concept. Each table in a relational database has a field in which the data value uniquely identifies the record. This field is called the primary key. The relationship is made between two tables by matching the values of the foreign key in one table with the values of the primary key in another.

Production Organization

A Salesforce organization that has live users accessing data.

Professional Edition

A Salesforce edition designed for businesses who need full-featured CRM functionality.

Q

Queue

A holding area for items before they are processed. Salesforce uses queues in a number of different features and technologies.

Query Locator

A parameter returned from the `query()` or `queryMore()` API call that specifies the index of the last result record that was returned.

Query String Parameter

A name-value pair that's included in a URL, typically after a '?' character. For example:

```
https://yourInstance.salesforce.com/001/e?name=value
```

R

Record

A single instance of a Salesforce object. For example, "John Jones" might be the name of a contact record.

Record Name

A standard field on all Salesforce objects. Whenever a record name is displayed in a Lightning Platform application, the value is represented as a link to a detail view of the record. A record name can be either free-form text or an autonumber field. `Record Name` does not have to be a unique value.

Record Type

A record type is a field available for certain records that can include some or all of the standard and custom picklist values for that record. You can associate record types with profiles to make only the included picklist values available to users with that profile.

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.

Recycle Bin

A page that lets you view and restore deleted information. Access the Recycle Bin either by using the link in the sidebar in Salesforce Classic or from the App Launcher in Lightning Experience.

Related Object

Objects chosen by an administrator to display in the Agent console's mini view when records of a particular type are shown in the console's detail view. For example, when a case is in the detail view, an administrator can choose to display an associated account, contact, or asset in the mini view.

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.

Report Type

A *report type* defines the set of records and fields available to a report based on the relationships between a primary object and its related objects. Reports display only records that meet the criteria defined in the report type. Salesforce provides a set of pre-defined standard report types; administrators can create custom report types as well.

Role Hierarchy

A record-level security setting that defines different levels of users such that users at higher levels can view and edit information owned by or shared with users beneath them in the role hierarchy, regardless of the organization-wide sharing model settings.

Roll-Up Summary Field

A field type that automatically provides aggregate values from child records in a master-detail relationship.

Running User


Each dashboard has a *running user*, whose security settings determine which data to display in a dashboard. If the running user is a specific user, all dashboard viewers see data based on the security settings of that user—regardless of their own personal security settings. For dynamic dashboards, you can set the running user to be the logged-in user, so that each user sees the dashboard according to his or her own access level.

S

SaaS

See Software as a Service (SaaS).

S-Control

 **Note:** S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

Custom Web content for use in custom links. Custom s-controls can contain any type of content that you can display in a browser, for example a Java applet, an Active-X control, an Excel file, or a custom HTML Web form.

Salesforce Record ID

A unique 15- or 18-character alphanumeric string that identifies a single record in Salesforce.

Salesforce SOA (Service-Oriented Architecture)

A powerful capability of Lightning Platform that allows you to make calls to external Web services from within Apex.

Sandbox

A nearly identical copy of a Salesforce production organization for development, testing, and training. The content and size of a sandbox varies depending on the type of sandbox and the edition of the production organization associated with the sandbox.

Search Layout

The organization of fields included in search results, in lookup dialogs, and in the key lists on tab home pages.

Search Phrase

Search phrases are queries that users enter when searching on www.google.com.

Semi-Join

A semi-join is a subquery on another object in an `IN` clause in a SOQL query. You can use semi-joins to create advanced queries, such as getting all contacts for accounts that have an opportunity with a particular record type. See also Anti-Join.

Session ID

An authentication token that is returned when a user successfully logs in to Salesforce. The Session ID prevents a user from having to log in again every time they want to perform another action in Salesforce. Different from a record ID or Salesforce ID, which are terms for the unique ID of a Salesforce record.

Session Timeout

The time after login before a user is automatically logged out. Sessions expire automatically after a predetermined length of inactivity, which can be configured in Salesforce from Setup by clicking **Security Controls**. The default is 120 minutes (two hours). The inactivity timer is reset to zero if a user takes an action in the web interface or makes an API call.

Setup

A menu where administrators can customize and define organization settings and Lightning Platform apps. Depending on your organization's user interface settings, Setup may be a link in the user interface header or in the dropdown list under your name.

Sharing

Allowing other users to view or edit information you own. There are different ways to share data:

- **Sharing Model**—defines the default organization-wide access levels that users have to each other's information and whether to use the hierarchies when determining access to data.
- **Role Hierarchy**—defines different levels of users such that users at higher levels can view and edit information owned by or shared with users beneath them in the role hierarchy, regardless of the organization-wide sharing model settings.
- **Sharing Rules**—allow an administrator to specify that all information created by users within a given group or role is automatically shared to the members of another group or role.
- **Manual Sharing**—allows individual users to share records with other users or groups.
- **Apex-Managed Sharing**—enables developers to programmatically manipulate sharing to support their application's behavior. See Apex-Managed Sharing.

Sharing Model

Behavior defined by your administrator that determines default access by users to different types of records.


Sharing Rule

Type of default sharing created by administrators. Allows users in a specified group or role to have access to all information created by users within a given group or role.

Sites

Salesforce Sites enables you to create public websites and applications that are directly integrated with your Salesforce organization—without requiring users to log in with a username and password.

Snippet

 **Note:** S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

A type of s-control that is designed to be included in other s-controls. Similar to a helper method that is used by other methods in a piece of code, a snippet allows you to maintain a single copy of HTML or JavaScript that you can reuse in multiple s-controls.

SOAP (Simple Object Access Protocol)

A protocol that defines a uniform way of passing XML-encoded data.

SOAP API

A SOAP-based Web services application programming interface that provides access to your Salesforce organization's information.

sObject

The abstract or parent object for all objects that can be stored in the Lightning Platform.

Software as a Service (SaaS)

A delivery model where a software application is hosted as a service and provided to customers via the Internet. The SaaS vendor takes responsibility for the daily maintenance, operation, and support of the application and each customer's data. The service alleviates the need for customers to install, configure, and maintain applications with their own hardware, software, and related IT resources. Services can be delivered using the SaaS model to any market segment.

SOQL (Salesforce Object Query Language)

A query language that allows you to construct simple but powerful query strings and to specify the criteria that selects data from the Lightning Platform database.

SOSL (Salesforce Object Search Language)

A query language that allows you to perform text-based searches using the Lightning Platform API.

Standard Object

A built-in object included with the Lightning Platform. You can also build custom objects to store information that is unique to your app.

Syndication Feeds

Give users the ability to subscribe to changes within Salesforce Sites and receive updates in external news readers.

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

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 Salesforce extensions for Visual Studio Code.

Translation Workbench

The Translation Workbench lets you specify languages you want to translate, assign translators to languages, create translations for customizations you've made to your Salesforce organization, and override labels and translations from managed packages. Everything from custom picklist values to custom fields can be translated so your global users can use Salesforce in their language.

Trigger

A piece of Apex that executes before or after records of a particular type are inserted, updated, or deleted from the database. Every trigger runs with a set of context variables that provide access to the records that caused the trigger to fire, and all triggers run in bulk mode—that is, they process several records at once, rather than just one record at a time.

Trigger Context Variable

Default variables that provide access to information about the trigger and the records that caused it to fire.

U

Unit Test

A unit is the smallest testable part of an application, usually a method. A unit test operates on that piece of code to make sure it works correctly. See also Test Method.

Unlimited Edition

Unlimited Edition is Salesforce's solution for maximizing your success and extending that success across the entire enterprise through the Lightning Platform.

Unmanaged Package

A package that cannot be upgraded or controlled by its developer.

URL (Uniform Resource Locator)

The global address of a website, document, or other resource on the Internet. For example, <https://salesforce.com>.

URL S-Control



Note: S-controls have been superseded by Visualforce pages. After March 2010 organizations that have never created s-controls, as well as new organizations, won't be allowed to create them. Existing s-controls will remain unaffected, and can still be edited.

An s-control that contains an external URL that hosts the HTML that should be rendered on a page. When saved this way, the HTML is hosted and run by an external website. URL s-controls are also called web controls.

V

Validation Rule

A rule that prevents a record from being saved if it does not meet the standards that are specified.

Visualforce

A simple, tag-based markup language that allows developers to easily define custom pages and components for apps built on the platform. Each tag corresponds to a coarse or fine-grained component, such as a section of a page, a related list, or a field. The components can either be controlled by the same logic that is used in standard Salesforce pages, or developers can associate their own logic with a controller written in Apex.

W

Web Control

See URL S-Control.

Web Links

See Custom Links.

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.

Web Services API

Term describing the original Salesforce Platform web services application programming interface (API) that provides access to your Salesforce org's information. See relevant developer guides for SOAP, REST, or Bulk APIs of interest.

WebService Method

An Apex class method or variable that external systems can use, like a mash-up with a third-party application. Web service methods must be defined in a global class.

Web Tab

A custom tab that allows your users to use external websites from within the application.

Automated Actions

Automated actions, such as email alerts, tasks, field updates, and outbound messages, can be triggered by a process, workflow rule, approval process, or milestone.

Workflow Action

A workflow action, such as an email alert, field update, outbound message, or task, fires when the conditions of a workflow rule are met.

Workflow Email Alert

A workflow action that sends an email when a workflow rule is triggered. Unlike workflow tasks, which can only be assigned to application users, workflow alerts can be sent to any user or contact, as long as they have a valid email address.

Workflow Field Update

A workflow action that changes the value of a particular field on a record when a workflow rule is triggered.

Workflow Outbound Message

A workflow action that sends data to an external Web service, such as another cloud computing application. Outbound messages are used primarily with composite apps.

Workflow Queue

A list of workflow actions that are scheduled to fire based on workflow rules that have one or more time-dependent workflow actions.

Workflow Rule

A workflow rule sets workflow actions into motion when its designated conditions are met. You can configure workflow actions to execute immediately when a record meets the conditions in your workflow rule, or set time triggers that execute the workflow actions on a specific day.

Workflow Task

A workflow action that assigns a task to an application user when a workflow rule is triggered.

Wrapper Class

A class that abstracts common functions such as logging in, managing sessions, and querying and batching records. A wrapper class makes an integration more straightforward to develop and maintain, keeps program logic in one place, and affords easy reuse across components. Examples of wrapper classes in Salesforce include the AJAX Toolkit, which is a JavaScript wrapper around the Salesforce SOAP API, wrapper classes such as `CCriticalSection` in the CTI Adapter for Salesforce CRM Call Center, or wrapper classes created as part of a client integration application that accesses Salesforce using the SOAP API.

WSC (Web Service Connector)

An XML-based Web service framework that consists of a Java implementation of a SOAP server. With WSC, developers can develop client applications in Java by using Java classes generated from Salesforce Enterprise WSDL or Partner WSDL.

WSDL (Web Services Description Language) File

An XML file that describes the format of messages you send and receive from a Web service. Your development environment's SOAP client uses the Salesforce Enterprise WSDL or Partner WSDL to communicate with Salesforce using the SOAP API.

X

No Glossary items for this entry.

Glossary

Y

No Glossary items for this entry.

Z

No Glossary items for this entry.