Version 20.0: Winter '11

Metadata API Developer's Guide

Last updated: November 6, 2010

Table of Contents

Getting Started.....5

Chapter 1: Understanding the Metadata API.....5

Supported Salesforce.com Editions.....6

Development Platforms.....6

Standards Compliance.....6

Metadata API Support Policy.....7

What's New in Version 20.0.....7

Related Resources.....8

Chapter 2: Quick Start.....9

Prerequisites.....9

Step 1: Create a Development Environment.....9

Step 2: Generate or Obtain the Web Service WSDLs.....10

Step 3: Import the WSDL Files Into Your Development Platform.....10

Step 4: Walk Through the Sample Code.....11

Using the Metadata API.....25

Chapter 3: Deploying and Retrieving Metadata.....25

Working with the Zip File.....25

Sample package.xml Manifest Files.....29

Running Tests in a Deployment.....31

Chapter 4: CRUD-Based Metadata Development.....33

Reference.....34

Chapter 5: File-Based Calls.....34

deploy().....34

retrieve().....42

Chapter 6: CRUD-Based Calls.....51

create().....51

delete().....52

update().....54

Chapter 7: Utility Calls.....57

checkStatus().....57

describeMetadata().....58

listMetadata().....59

Chapter 8: Result Objects.....61

AsyncResult.....61

DeployResult..63
DescribeMetadataResult..67
RetrieveResult..68

Chapter 9: Metadata Types..71
Field Data Types..71
Supported Calls..72
AnalyticSnapshot..72
ArticleType..74
ApexClass..80
ApexComponent..83
ApexPage..84
ApexTrigger..85
CustomApplication..87
CustomLabels..88
CustomObject..89
CustomObjectTranslation..125
CustomPageWebLink..131
CustomSite..134
CustomTab..137
Dashboard..139
DataCategoryGroup..148
Document..153
EmailTemplate..155
EntitlementTemplate..157
Folder..158
HomePageComponent..160
HomePageLayout..161
Layout..162
Letterhead..168
Metadata..170
MetadataWithContent..171
Portal..172
Profile..174
RemoteSiteSetting..182
Report..183
ReportType..199
Scontrol..202
StaticResource..204
Translations..205
Workflow..210

Chapter 10: Error Handling..221
Error Handling for Session Expiration 221

Glossary..........222

Index..........237

Chapter 1

Understanding the Metadata API

Use the Metadata API to retrieve, deploy, create, update or delete customization information, such as custom object definitions and page layouts, for your organization. This API is intended for managing customizations and for building tools that can manage the metadata model, not the data itself. To create, retrieve, update or delete *records*, such as accounts or leads, use the Web services API to manage your data.

The easiest way to access the functionality in the Metadata API is to use the Force.com IDE or Force.com Migration Tool. These tools are built on top of the Metadata API and use the standard Eclipse and Ant tools respectively to simplify the task of working with the Metadata API. Built on the Eclipse platform, the Force.com IDE provides a comfortable environment for programmers familiar with integrated development environments, allowing you to code, compile, test, and deploy all from within the IDE itself. The Force.com Migration Tool is ideal if you want to use a script or a command-line utility for moving metadata between a local directory and a Salesforce.com organization. For more information about the Force.com IDE or Force.com Migration Tool, see developer.force.com.

The underlying calls of the Metadata API have been exposed for you to use directly, if you prefer to build your own client applications. This guide gives you more information about working directly with the Metadata API.

You can use the *asynchronous* Metadata API to manage *setup* and customization information (metadata) for your organizations. For example:

- Export the customizations in your organization as XML metadata files. See Working with the Zip File and `retrieve()`.
- Migrate configuration changes between organizations. See `deploy()` and `retrieve()`.
- Modify existing customizations in your organization using XML metadata files. See `deploy()` and `retrieve()`.
- Manage customizations in your organization programmatically. See CRUD-Based Metadata Development, `create()`, `update()`, and `delete()`.

You can modify metadata in *test* organizations on Developer Edition or *sandbox*, and then deploy tested changes to *production* organizations on Enterprise Edition or Unlimited Editions. You can also create scripts to populate a new organization with your custom objects, custom fields, and other *components*.

Tip: This guide is available in HTML and PDF formats at `http://developer.force.com`.

Supported Salesforce.com Editions

To use the Metadata API, your organization must use Enterprise Edition, Unlimited Edition, Free Edition, or Developer Edition. If you are an existing Salesforce.com customer and want to upgrade to either Enterprise or Unlimited Edition, contact your account representative.

It is strongly recommended that you use a sandbox, which is an exact replica of your production organization. Enterprise, Free, and Unlimited Editions come with a free Developer sandbox. For more information, see `http://www.salesforce.com/platform/cloud-infrastructure/sandbox.jsp`.

Alternatively, you can use a Developer Edition organization, which provides access to all of the features available with Enterprise Edition, but is limited by the number of users and the amount of storage space. A Developer Edition organization is not a copy of your production organization, but it provides an environment where you can build and test your solutions without affecting your organization's data. Developer Edition accounts are available for free at `http://developer.force.com/join`.

Note: A metadata component must be visible in the organization for the Metadata API to act on it. Also, a user must be assigned to a profile with the "API Enabled" permission to have access to metadata components.

Development Platforms

The Metadata API supports both file-based and CRUD-based development.

File-Based Development

The declarative or file-based asynchronous Metadata API `deploy()` and `retrieve()` calls deploy or retrieve a `.zip` file that holds components in a set of folders, and a manifest file named `package.xml`. For more information, see Deploying and Retrieving Metadata on page 25. The easiest way to access the file-based functionality is to use the Force.com IDE or Force.com Migration Tool.

CRUD-Based Development

The CRUD-based asynchronous Metadata API calls `create()`, `update()`, and `delete()` act upon the metadata components in a manner similar to the way synchronous API calls in the *enterprise WSDL* act upon objects. For more information about the enterprise WSDL, see the *Web Services API Developer's Guide*.

Note: CRUD (create, read, update, delete) implies that there is a read call, but there is no equivalent read call for CRUD-based development. If you want to read your metadata, you should use the `retrieve()` call, described in File-Based Development on page 5.

Use the `create()`, `update()`, and `delete()` calls with the utility call `checkStatus()`. For more information, see CRUD-Based Metadata Development on page 33 and Quick Start on page 9.

Standards Compliance

The Metadata API is implemented to comply with the following specifications:

Standard Name	Website
Simple Object Access Protocol (SOAP) 1.1	`http://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`

Metadata API Support Policy

Salesforce.com support previous versions of the Metadata API. However, your new client applications should use the most recent version of the Force.com Metadata API WSDL file to fully exploit the benefits of richer features and greater efficiency.

Backward Compatibility

Salesforce.com strives to make backward compatibility easy when using the Force.com platform.

Each new Salesforce.com release consists of two components:

- A new release of platform software that resides on salesforce.com systems
- A new version of the API

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

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

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

For example, an application written using API version 9.0 which shipped with the Spring ’07 release will continue to work with API version 9.0 on the Summer ’07 release and on future releases beyond that. However, that same application may not work with API version 10.0 without modifications to the application.

API End-of-Life

Salesforce.com 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 may cease to be supported.

When an API version is to be deprecated, advance end-of-life notice will be given at least one year before support for the API version is ended. Salesforce.com will directly notify customers using API versions planned for deprecation.

What’s New in Version 20.0

The following metadata fields have been added or changed in Metadata API version 20.0:

Metadata Type or Related Object	Field	Change	Description
CustomObject	`recordTypeTrackFeedHistory`	New	Indicates whether the record type is enabled for feed tracking (`true`) or not (`false`).
CustomObject	`recordTypeTrackHistory`	New	Indicates whether history tracking is enabled for this record type (`true`) or not (`false`).
CustomSite	`siteRedirectMappings`	New	An array of all URL redirect rules set for your site.
Dashboard	`dashboardType`	Updated	A new valid value has been added: `MyTeamUser`—Managers can choose to view the dashboard from the point of view of their subordinates in the role hierarchy.
Dashboard	`drillToDetailEnabled`	New	When enabled, users are taken to the record detail page when they click a record name, record owner, or feed post in a table or chart. When set to `true` users can click axis and legend values, chart elements, and table entries. The `drillDownUrl` and `drillEnabled` fields override this field.

Related Resources

The salesforce.com 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 `https://wiki.developerforce.com/index.php/Getting_Started` for more information, or visit http://developer.force.com/join to sign up for a free Developer Edition account.

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

- Developer Force provides a wealth of information for developers.
- Salesforce.com for information about the Salesforce.com application.
- Force.com AppExchange for access to apps created for Salesforce.com.
- Salesforce.com Community for services to ensure Salesforce.com customer success.

Chapter 2

Quick Start

The easiest way to access the functionality in the Metadata API is to use the Force.com IDE or Force.com Migration Tool. These tools are built on top of the Metadata API and use the standard Eclipse and Ant tools respectively to simplify the task of working with the Metadata API. Built on the Eclipse platform, the Force.com IDE provides a comfortable environment for programmers familiar with integrated development environments, allowing you to code, compile, test, and deploy all from within the IDE itself. The Force.com Migration Tool is ideal if you want to use a script or a command-line utility for moving metadata between a local directory and a Salesforce.com organization. For more information about the Force.com IDE or Force.com Migration Tool, see developer.force.com.

However, the underlying calls of the Metadata API have been exposed for you to use directly, if you prefer to build your own client applications. This quick start gives you all the information you need to start writing applications that directly use the metadata API to manage customizations for your organization. It shows you how to get started with file-based development and CRUD-Based Development.

Prerequisites

Before you can start using the Metadata API, you need to do some background work:

- Identify a user that has a profile with the "API Enabled" and "Modify All Data" permissions. These permissions are required to access Metadata API calls.
- Install a SOAP client. The Metadata API works with current SOAP development environments, including, but not limited to, Visual Studio .NET 2005, and Apache Axis. In this document, we provide examples in Java. The Java examples are based on Apache Axis 1.3 and JDK 5.0 (Java 2 Platform Standard Edition Development Kit 5.0). For more information about Apache Axis 1.3, go to `http://ws.apache.org/axis/`. To see a complete list of compatible development platforms and more sample code, go to developer.force.com.

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

Step 1: Create a Development Environment

It is strongly recommended that you use a sandbox, which is an exact replica of your production organization. Enterprise, Free, and Unlimited Editions come with a free Developer sandbox. For more information, see `http://www.salesforce.com/platform/cloud-infrastructure/sandbox.jsp`.

Alternatively, you can use a Developer Edition organization, which provides access to all of the features available with Enterprise Edition, but is limited by the number of users and the amount of storage space. A Developer Edition organization is not a copy of your production organization, but it provides an environment where you can build and test your solutions without affecting your organization's data. Developer Edition accounts are available for free at `http://developer.force.com/join`.

Step 2: Generate or Obtain the Web Service WSDLs

To access Metadata API calls, 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 stub code to access the Web service it defines. You can either obtain the WSDL file from your organization's Salesforce.com administrator or you can generate it yourself if you have access to the WSDL download page in the Salesforce.com user interface. For more information about WSDL, see `http://www.w3.org/TR/wsdl`.

Before you can access Metadata API calls, you must authenticate to use the Web service using the `login()` call, which is defined in the enterprise WSDL and the partner WSDL. Therefore, you must also obtain either of these WSDLs. For more information about the enterprise WSDL and the partner WSDL, see Step 2: Generate or Obtain the Web Service WSDL in the *Web Services API Developer's Guide*.

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 the Salesforce.com platform. (The System Administrator profile has this permission.)

The sample code in Step 4: Walk Through the Sample Code uses the enterprise WSDL, though the partner WSDL works equally well.

To generate the metadata and enterprise WSDL files for your organization:

1. Log in to your Salesforce.com account. You must log in as an administrator or as a user who has the "Modify All Data" permission.
2. Click ***Your Name*** ➤ **Setup** ➤ **Develop** ➤ **API**.
3. Click the **Download Metadata WSDL** link and save the XML WSDL file to your file system.
4. Click the **Download Enterprise WSDL** link and save the XML WSDL file to your file system.

Step 3: Import the WSDL Files Into Your Development Platform

Once you have the WSDL files, import them into your development platform so that your development environment can generate the necessary objects for use in building client Web service applications. This section provides sample instructions for Apache Axis. For instructions about other development platforms, see your platform's product documentation.

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

Instructions for Java Environments (Apache Axis)

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 Apache Axis, use the WSDL2Java utility.

Note: Before you run WSDL2Java, you must have Axis installed on your system and all of its component JAR files must be referenced in your classpath.

The basic syntax for WSDL2Java is:

```
java -classpath pathToJAR/Filename org.apache.axis.wsdl.WSDL2Java -a pathToWsdl/WsdlFilename
```

The `-a` switch generates code for all elements, referenced or not, which may be necessary depending on your WSDL. For more information, see the WSDL2Java documentation.

If you have JAR files in more than one location, list them with a semicolon separating the files. For example, if the Axis JAR files are installed in `C:\axis-1.3`, and the WSDL is named `my_metadata.wsdl` and is stored in `C:\mywsdls`, use the following command to generate the Java stub files:

Note: The following command has line breaks for formatting purposes only.

```
java -classpath c:\axis-1.3\lib\axis.jar;c:\axis-1.3\lib\axis-ant.jar;
c:\axis-1.3\lib\axis-schema.jar;c:\axis-1.3\lib\commons-discovery-0.2.jar;
c:\axis-1.3\lib\commons-logging-1.0.4.jar;c:\axis-1.3\lib\jaxrpc.jar;
c:\axis-1.3\lib\log4j-1.2.8.jar;c:\axis-1.3\lib\saaj.jar;c:\axis-1.3\lib\wsdl4j-1.5.2.jar;
c:\axis-1.3\mail.jar;c:\axis-1.3\activation.jar;c:\axis-1.3\wsdl4j.jar;
org.apache.axis.wsdl.WSDL2Java -a C:\mywsdls\my_metadata.wsdl
```

To generate stub files for the enterprise WSDL, use a similar command:

```
java -classpath
c:\axis-1.3\lib\axis.jar;c:\axis-1.3\lib\axis-ant.jar;c:\axis-1.3\lib\axis-schema.jar;
c:\axis-1.3\lib\commons-discovery-0.2.jar;c:\axis-1.3\lib\commons-logging-1.0.4.jar;
c:\axis-1.3\lib\jaxrpc.jar;c:\axis-1.3\lib\log4j-1.2.8.jar;c:\axis-1.3\lib\saaj.jar;
c:\axis-1.3\lib\wsdl4j-1.5.2.jar;c:\axis-1.3\mail.jar;c:\axis-1.3\activation.jar;
c:\axis-1.3\wsdl4j.jar;  org.apache.axis.wsdl.WSDL2Java -a C:\mywsdls\my_enterprise.wsdl
```

These commands generate a set of folders and Java source code files in the same directory in which they were run. After these files are compiled, they can be included in your Java programs for use in creating client applications.

For most Java development environments, you can use wizard-based tools for this process instead of the command line. For more information about using WSDL2Java, see `http://ws.apache.org/axis/java/reference.html`. For more information about using WSDL2Java with Force.com, visit the message boards at `http://www.salesforce.com/developer/boards.jsp`.

Step 4: Walk Through the Sample Code

Once you have imported the WSDL files, you can begin building client applications that use the Metadata API. The samples listed below are a good starting point for writing your own code.

- Java Sample Code for File-Based Development on page 9
- Java Sample Code for CRUD-Based Development on page 9

Java Sample Code for File-Based Development

This section walks through a sample Java client application that uses file-based development calls. The purpose of this sample application is to show the required steps for authentication using the `login()` call and to demonstrate the invocation and subsequent handling of several Metadata API calls, including `retrieve()` and `deploy()`. The sample application performs the following main tasks:

1. Prompts the user for their Salesforce.com username and password.
2. Calls `login()` which is part of the enterprise WSDL and, if the login succeeds:
   - Sets the returned `sessionId` into the session header for the metadata SOAP binding. This is required for session authentication on subsequent Metadata API calls.
   - Resets the endpoint to the returned `metadataServerUrl`, which is the target of subsequent Metadata API calls.

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

3. Prompts the user to execute a `retrieve()` and `deploy()` call. The user can execute `retrieve()` or `deploy()` calls until they choose the exit option.

The `retrieve()` and `deploy()` calls both operate on a zip file named `components.zip`. The `retrieve()` call retrieves components from your organization into `components.zip`, and the `deploy()` call deploys the components in `components.zip` to your organization. If you save the sample to your computer and execute it, you should run the retrieve option first so that you have a `components.zip` file that you can subsequently deploy.

The `retrieve()` call uses a manifest file to determine the components to retrieve from your organization. A sample `package.xml` manifest file is listed below. For more details on the manifest file structure, see Working with the Zip File on page 25. For the purposes of this sample, it is sufficient to know that this manifest file retrieves all custom objects, custom tabs, and page layouts.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>CustomObject</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomTab</name>
    </types>
    <types>
        <members>*</members>
        <name>Layout</name>
    </types>
    <version>20.0</version>
</Package>
```

Note the error handling code that follows each API call.

Note: This sample was created using Apache Axis. The WSDL2Java utility generates a `_package` class, even though the metadata type is defined as `Package` in the Metadata WSDL. Other SOAP clients may generate a different name for the `_package` class.

```
package com.doc.samples;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.channels.Channels;
import java.nio.channels.FileChannel;
import java.nio.channels.ReadableByteChannel;
import java.nio.channels.WritableByteChannel;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.List;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.rpc.ServiceException;

import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import com.sforce.soap._2006._04.metadata.AsyncRequestState;
import com.sforce.soap._2006._04.metadata.AsyncResult;
import com.sforce.soap._2006._04.metadata.CodeCoverageWarning;
import com.sforce.soap._2006._04.metadata.DeployMessage;
import com.sforce.soap._2006._04.metadata.DeployOptions;
import com.sforce.soap._2006._04.metadata.DeployResult;
import com.sforce.soap._2006._04.metadata.MetadataBindingStub;
import com.sforce.soap._2006._04.metadata.MetadataServiceLocator;
import com.sforce.soap._2006._04.metadata.PackageTypeMembers;
import com.sforce.soap._2006._04.metadata.RetrieveMessage;
import com.sforce.soap._2006._04.metadata.RetrieveRequest;
import com.sforce.soap._2006._04.metadata.RetrieveResult;
import com.sforce.soap._2006._04.metadata.RunTestFailure;
import com.sforce.soap._2006._04.metadata.RunTestsResult;
// Note that Axis generates a _package class, even though it is defined as Package
// in the WSDL. Other SOAP clients may generate a different name for the _package class.
import com.sforce.soap._2006._04.metadata._package;
import com.sforce.soap.enterprise.LoginResult;
import com.sforce.soap.enterprise.SessionHeader;
import com.sforce.soap.enterprise.SforceServiceLocator;
import com.sforce.soap.enterprise.SoapBindingStub;
import com.sforce.soap.enterprise.fault.ExceptionCode;
import com.sforce.soap.enterprise.fault.LoginFault;

public class DeployRetrieveFileBased
{
    // binding for the Enterprise WSDL used for login() call
    private SoapBindingStub binding;
    // binding for the metadata WSDL used for create() and checkStatus() calls
    private MetadataBindingStub metadatabinding;

    static BufferedReader rdr = new BufferedReader(new InputStreamReader(System.in));

    private static final String ZIP_FILE = "components.zip";

    // manifest file that controls which components get retrieved
    private static final String MANIFEST_FILE = "package.xml";

    private static final double API_VERSION = 15.0;

    // one second in milliseconds
    private static final long ONE_SECOND = 1000;
    // maximum number of attempts to deploy the zip file
    private static final int MAX_NUM_POLL_REQUESTS = 50;

    public static void main(String[] args) throws ServiceException, Exception {
        DeployRetrieveFileBased sample = new DeployRetrieveFileBased();
        sample.run();
    }
```

```
private void run() throws ServiceException, Exception {
    if (login()) {
        getUserInput("SUCCESSFUL LOGIN! Hit the enter key to continue.");
        // Show the options.
        showMenu();
        String choice = getUserInput("");
        // Show the options to retrieve or deploy until user exits
        while (choice != null && !choice.equals("99")) {
            try {
                if (choice.length() == 1 || choice.length() == 2) {
                    switch (new Integer(choice).intValue()) {
                    case 1:
                        retrieveZip();
                        break;
                    case 2:
                        deployZip();
                        break;
                    }
                }
                // show the menu again
                showMenu();

            } catch (Exception e) {
                System.out.println("An unexpected error has occurred: "
                        + e.getMessage());
                e.printStackTrace();
            }
            //wait for the user input.
            choice = getUserInput("");
        }
    }
}

private void deployZip()
    throws RemoteException, Exception
{
    byte zipBytes[] = readZipFile();
    DeployOptions deployOptions = new DeployOptions();
    deployOptions.setPerformRetrieve(false);
    deployOptions.setRollbackOnError(true);
    AsyncResult asyncResult = metadatabinding.deploy(zipBytes, deployOptions);

    // Wait for the deploy to complete
    int poll = 0;
    long waitTimeMilliSecs = ONE_SECOND;
    while (!asyncResult.isDone()) {
        Thread.sleep(waitTimeMilliSecs);
        // double the wait time for the next iteration
        waitTimeMilliSecs *= 2;
        if (poll++ > MAX_NUM_POLL_REQUESTS) {
            throw new Exception("Request timed out. If this is a large set " +
                    "of metadata components, check that the time allowed by " +
                    "MAX_NUM_POLL_REQUESTS is sufficient.");
        }
        asyncResult = metadatabinding.checkStatus(
                new String[] {asyncResult.getId()})[0];
        System.out.println("Status is: " + asyncResult.getState());
    }

    if (asyncResult.getState() != AsyncRequestState.Completed) {
        throw new Exception(asyncResult.getStatusCode() + " msg: " +
                asyncResult.getMessage());
    }

    DeployResult result = metadatabinding.checkDeployStatus(asyncResult.getId());
    if (!result.isSuccess()) {
        printErrors(result);
        throw new Exception("The files were not successfully deployed");
    }

    System.out.println("The file " + ZIP_FILE + " was successfully deployed\n");
}

/**
 * Read in the zip file contents into a byte array.
 * @return byte[]
 * @throws Exception - if cannot find the zip file to deploy
 */
private byte[] readZipFile()
    throws Exception
{
    // We assume here that you have a deploy.zip file.
    // See the retrieve sample for how to retrieve a zip file.
    File deployZip = new File(ZIP_FILE);
    if (!deployZip.exists() || !deployZip.isFile())
        throw new Exception("Cannot find the zip file to deploy. Looking for " +
                deployZip.getAbsolutePath());

    FileInputStream fos = new FileInputStream(deployZip);
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    int readbyte = -1;
    while ((readbyte = fos.read()) != -1)  {
        bos.write(readbyte);
    }
    fos.close();
    bos.close();
    return bos.toByteArray();
}


/**
 * Print out any errors, if any, related to the deploy.
 * @param result - DeployResult
 */
private void printErrors(DeployResult result)
{
    DeployMessage messages[] = result.getMessages();
    StringBuilder buf = new StringBuilder("Failures:\n");
    for (DeployMessage message : messages) {
        if (!message.isSuccess()) {
            String loc = (message.getLineNumber() == null ? "" :
                ("(" + message.getLineNumber() + "," +
                        message.getColumnNumber() + ")"));
            if (loc.length() == 0
                    && !message.getFileName().equals(message.getFullName())) {
                loc = "(" + message.getFullName() + ")";
            }
            buf.append(message.getFileName() + loc + ":" +
                    message.getProblem()).append('\n');
        }
    }
    RunTestsResult rtr = result.getRunTestResult();
    if (rtr.getFailures() != null) {
        for (RunTestFailure failure : rtr.getFailures()) {
            String n = (failure.getNamespace() == null ? "" :
                (failure.getNamespace() + ".")) + failure.getName();
            buf.append("Test failure, method: " + n + "." +
                    failure.getMethodName() + " -- " +
                    failure.getMessage() + " stack " +
                    failure.getStackTrace() + "\n\n");
        }
    }
    if (rtr.getCodeCoverageWarnings() != null) {
        for (CodeCoverageWarning ccw : rtr.getCodeCoverageWarnings()) {
            buf.append("Code coverage issue");
            if (ccw.getName() != null) {
                String n = (ccw.getNamespace() == null ? "" :
                    (ccw.getNamespace() + ".")) + ccw.getName();
                buf.append(", class: " + n);
            }
            buf.append(" -- " + ccw.getMessage() + "\n");
        }
    }

    System.out.println(buf.toString());
}


private void retrieveZip() throws RemoteException, Exception
{
    RetrieveRequest retrieveRequest = new RetrieveRequest();
    retrieveRequest.setApiVersion(API_VERSION);
    setUnpackaged(retrieveRequest);

    AsyncResult asyncResult = metadatabinding.retrieve(retrieveRequest);
    // Wait for the retrieve to complete
    int poll = 0;
    long waitTimeMilliSecs = ONE_SECOND;
    while (!asyncResult.isDone()) {
        Thread.sleep(waitTimeMilliSecs);
        // double the wait time for the next iteration
        waitTimeMilliSecs *= 2;
        if (poll++ > MAX_NUM_POLL_REQUESTS) {
            throw new Exception("Request timed out.  If this is a large set " +
                    "of metadata components, check that the time allowed " +
                    "by MAX_NUM_POLL_REQUESTS is sufficient.");
        }
        asyncResult = metadatabinding.checkStatus(
                new String[] {asyncResult.getId()})[0];
        System.out.println("Status is: " + asyncResult.getState());
    }

    if (asyncResult.getState() != AsyncRequestState.Completed) {
        throw new Exception(asyncResult.getStatusCode() + " msg: " +
                asyncResult.getMessage());
    }

    RetrieveResult result = metadatabinding.checkRetrieveStatus(asyncResult.getId());

    // Print out any warning messages
    StringBuilder buf = new StringBuilder();
    if (result.getMessages() != null) {
        for (RetrieveMessage rm : result.getMessages()) {
            buf.append(rm.getFileName() + " - " + rm.getProblem());
        }
    }
    if (buf.length() > 0) {
        System.out.println("Retrieve warnings:\n" + buf);
    }

    // Write the zip to the file system
    System.out.println("Writing results to zip file");
    ByteArrayInputStream bais = new ByteArrayInputStream(result.getZipFile());
    File resultsFile = new File(ZIP_FILE);
    FileOutputStream os = new FileOutputStream(resultsFile);
    try {
        ReadableByteChannel src = Channels.newChannel(bais);
        FileChannel dest = os.getChannel();
        copy(src, dest);
        System.out.println("Results written to "
                + resultsFile.getAbsolutePath() + "\n");
    }
    finally {
        os.close();
    }

}

/**
 * Helper method to copy from a readable channel to a writable channel,
 * using an in-memory buffer.
 */
private void copy(ReadableByteChannel src, WritableByteChannel dest)
    throws IOException
{
    // use an in-memory byte buffer
    ByteBuffer buffer = ByteBuffer.allocate(8092);
    while (src.read(buffer) != -1) {
        buffer.flip();
        while(buffer.hasRemaining()) {
            dest.write(buffer);
        }
        buffer.clear();
    }
}

private void setUnpackaged(RetrieveRequest request) throws Exception
{
    // Edit the path, if necessary, if your package.xml file is located elsewhere
    File unpackedManifest = new File(MANIFEST_FILE);
    System.out.println("Manifest file: " + unpackedManifest.getAbsolutePath());

    if (!unpackedManifest.exists() || !unpackedManifest.isFile())
        throw new Exception("Should provide a valid retrieve manifest " +
                "for unpackaged content. " +
                "Looking for " + unpackedManifest.getAbsolutePath());

    // Note that we populate the _package object by parsing a manifest file here.
    // You could populate the _package based on any source for your
    // particular application.
    _package p = parsePackage(unpackedManifest);
    request.setUnpackaged(p);
}

private _package parsePackage(File file) throws Exception {
    try {
        InputStream is = new FileInputStream(file);
        List<PackageTypeMembers> pd = new ArrayList<PackageTypeMembers>();
        DocumentBuilder db =
            DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Element d = db.parse(is).getDocumentElement();
        for (Node c = d.getFirstChild(); c != null; c = c.getNextSibling()) {
            if (c instanceof Element) {
                Element ce = (Element)c;
                //
                NodeList namee = ce.getElementsByTagName("name");
                if (namee.getLength() == 0) {
                    // not
                    continue;
                }
                String name = namee.item(0).getTextContent();
                NodeList m = ce.getElementsByTagName("members");
                List<String> members = new ArrayList<String>();
                for (int i = 0; i < m.getLength(); i++) {
                    Node mm = m.item(i);
                    members.add(mm.getTextContent());
                }
                PackageTypeMembers pdi = new PackageTypeMembers();
                pdi.setName(name);
                pdi.setMembers(members.toArray(new String[members.size()]));
                pd.add(pdi);
            }
        }
        _package r = new _package();
        r.setTypes(pd.toArray(new PackageTypeMembers[pd.size()]));
        r.setVersion(API_VERSION + "");
        return r;
    } catch (ParserConfigurationException pce) {
        throw new Exception("Cannot create XML parser", pce);
    } catch (IOException ioe) {
        throw new Exception(ioe);
    } catch (SAXException se) {
        throw new Exception(se);
    }
}

/**
 * Utility method to present options to retrieve or deploy.
 * This method prints all the possible sample names to the console
 * so that the user can select a particular sample by entering the corresponding
 * number of the sample. Once the user enters the sample number, that particular
 * sample will be invoked and run.
 */
private void showMenu() {

    System.out.println(" 1: Retrieve");
    System.out.println(" 2: Deploy");
    System.out.println("99: Exit");
    System.out.println("   ");
    System.out.print("Enter 1 to retrieve, 2 to deploy, or 99 to exit: ");
}

/**
 * The login call is used to obtain a token from Salesforce.
 * This token must be passed to all other calls to provide
 * authentication.
 */
private boolean login() throws ServiceException {
    String userName = getUserInput("Enter username: ");
    String password = getUserInput("Enter password: ");
    /** Next, the sample client application initializes the binding stub.
     *
     * This is our main interface to the API for the Enterprise WSDL.
     * The getSoap method takes an optional parameter,
     * (a java.net.URL) which is the endpoint.
     * For the login call, the parameter always starts with
     * http(s)://login.salesforce.com. After logging in, the sample
     * client application changes the endpoint to the one specified
     * in the returned loginResult object.
     */
    binding = (SoapBindingStub) new SforceServiceLocator().getSoap();

    // Time out after a minute
    binding.setTimeout(60000);
    // Log in using the Enterprise WSDL binding
    LoginResult loginResult;
    try {
        System.out.println("LOGGING IN NOW....");
        loginResult = binding.login(userName, password);
    }
    catch (LoginFault ex) {

        // The LoginFault derives from AxisFault
        ExceptionCode exCode = ex.getExceptionCode();
        if (exCode == ExceptionCode.FUNCTIONALITY_NOT_ENABLED ||
            exCode == ExceptionCode.INVALID_CLIENT ||
            exCode == ExceptionCode.INVALID_LOGIN ||
            exCode == ExceptionCode.LOGIN_DURING_RESTRICTED_DOMAIN ||
            exCode == ExceptionCode.LOGIN_DURING_RESTRICTED_TIME ||
            exCode == ExceptionCode.ORG_LOCKED ||
            exCode == ExceptionCode.PASSWORD_LOCKOUT ||
            exCode == ExceptionCode.SERVER_UNAVAILABLE ||
            exCode == ExceptionCode.TRIAL_EXPIRED ||
            exCode == ExceptionCode.UNSUPPORTED_CLIENT) {
            System.out.println("Please be sure that you have a valid username " +
                    "and password.");
        } else {
            // Write the fault code to the console
            System.out.println(ex.getExceptionCode());
            // Write the fault message to the console
            System.out.println("An unexpected error has occurred." + ex.getMessage());
        }
        return false;
    } catch (Exception ex) {
        System.out.println("An unexpected error has occurred: " + ex.getMessage());
        ex.printStackTrace();
        return false;
    }
    // Check if the password has expired
    if (loginResult.isPasswordExpired()) {
        System.out.println("An error has occurred. Your password has expired.");
        return false;
    }

    /** Once the client application has logged in successfully, we use
     *  the results of the login call to reset the endpoint of the service
     *  to the virtual server instance that is servicing your organization.
     *  To do this, the client application sets the ENDPOINT_ADDRESS_PROPERTY
     *  of the binding object using the URL returned from the LoginResult. We
     *  use the metadata binding from this point forward as we are invoking
     *  calls in the metadata WSDL.
     */
    metadatabinding = (MetadataBindingStub)
            new MetadataServiceLocator().getMetadata();
    metadatabinding._setProperty(MetadataBindingStub.ENDPOINT_ADDRESS_PROPERTY,
            loginResult.getMetadataServerUrl());

    /** The sample client application now has an instance of the MetadataBindingStub
     *  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 the SoapBindingStub) that contains the valid sessionId
     *  for our login credentials. To do this, the sample client application
     *  creates a new SessionHeader object and set its sessionId property to the
     *  sessionId property from the LoginResult object.
     */
    // Create a new session header object and add the session id
    // from the login return object
    SessionHeader sh = new SessionHeader();
    sh.setSessionId(loginResult.getSessionId());
    /** Next, the sample client application calls the setHeader method of the
     *  SoapBindingStub to add the header to all subsequent method calls. This
     *  header will persist until the binding is destroyed or until the header
     *  is explicitly removed. The "SessionHeader" parameter is the name of the
     *  header to be added.
     */
    // set the session header for subsequent call authentication
    metadatabinding.setHeader(
        new MetadataServiceLocator().getServiceName().getNamespaceURI(),
            "SessionHeader", sh);
    // return true to indicate that we are logged in, pointed
    // at the right url and have our security token in place.
    return true;
}

//The sample client application retrieves the user's login credentials.
// Helper function for retrieving user input from the console
String getUserInput(String prompt) {
    System.out.print(prompt);
    try {
        return rdr.readLine();
    }
    catch (IOException ex) {
        return null;
    }
}
}
```

Java Sample Code for CRUD-Based Development

This section walks through a sample Java client application that uses CRUD-Based Development calls. The purpose of this sample application is to show the required steps for authentication using the `login()` call and to demonstrate the invocation and subsequent handling of several Metadata API calls, including `create()` and `checkStatus()`. This sample application performs the following main tasks:

1. Prompts the user for their Salesforce.com username and password.
2. Calls `login()` which is part of the enterprise WSDL and, if the login succeeds:
    - Sets the returned `sessionId` into the session header for the metadata SOAP binding. This is required for session authentication on subsequent Metadata API calls.
    - Resets the endpoint to the returned `metadataServerUrl`, which is the target of subsequent Metadata API calls.

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

3. Calls `create()` to create a new custom object.

    Salesforce.com returns an AsyncResult object for each component you tried to create. The AsyncResult object is updated with status information as the operation moves from a queue to completed or error state.
4. Calls `checkStatus()` in a loop until the status value in AsyncResult indicates that the create operation is completed. Start with a wait time of one second between iterations of `checkStatus()` calls, and double the wait time each time you make a subsequent call.

Note the error handling code that follows each API call.

```
package com.doc.samples;

import java.io.*;
import javax.xml.rpc.ServiceException;

import com.sforce.soap.enterprise.*;
import com.sforce.soap.enterprise.fault.ExceptionCode;
import com.sforce.soap.enterprise.fault.LoginFault;

import com.sforce.soap._2006._04.metadata.MetadataBindingStub;
import com.sforce.soap._2006._04.metadata.MetadataServiceLocator;
import com.sforce.soap._2006._04.metadata.AsyncResult;
import com.sforce.soap._2006._04.metadata.CustomField;
import com.sforce.soap._2006._04.metadata.CustomObject;
import com.sforce.soap._2006._04.metadata.DeploymentStatus;
import com.sforce.soap._2006._04.metadata.FieldType;
import com.sforce.soap._2006._04.metadata.SharingModel;
```

```
/**
 * Title: Sample that logs in and creates a custom object.
 *
 * Description: Console application illustrating login, session management,
 * and server redirection.
 *
 * Assumptions:
 * 1. The Enterprise WSDL has been downloaded and sample stub code has been generated
 *      by a tool like WSDL2Java. The Enterprise or Partner WSDL must be used to
 *      use the login() call for authentication. In this case, we use the Enterprise WSDL.
 * 2. The Metadata WSDL has been downloaded and sample stub code has been generated
 *      by a tool like WSDL2Java.
 *
 */
public class CreateSample {
    // binding for the Enterprise WSDL used for login() call
    private SoapBindingStub binding;
    // binding for the metadata WSDL used for create() and checkStatus() calls
    private MetadataBindingStub metadatabinding;

    static BufferedReader rdr = new BufferedReader(new InputStreamReader(System.in));

    // one second in milliseconds
    private static final long ONE_SECOND = 1000;

    public CreateSample() {
    }

    public static void main(String[] args) throws ServiceException {
        CreateSample samples1 = new CreateSample();
        samples1.run();
    }

    //The sample client application retrieves the user's login credentials.
    // Helper function for retrieving user input from the console
    String getUserInput(String prompt) {
        System.out.print(prompt);
        try {
            return rdr.readLine();
        }
        catch (IOException ex) {
            return null;
        }
    }

    /**
     * The login call is used to obtain a token from Salesforce.
     * This token must be passed to all other calls to provide
     * authentication.
     */
    private boolean login() throws ServiceException {
        String userName = getUserInput("Enter username: ");
        String password = getUserInput("Enter password: ");
        /** Next, the sample client application initializes the binding stub.
         *
         * This is our main interface to the API for the Enterprise WSDL.
         * The getSoap method takes an optional parameter,
         * (a java.net.URL) which is the endpoint.
         * For the login call, the parameter always starts with
         * http(s)://login.salesforce.com. After logging in, the sample
         * client application changes the endpoint to the one specified
         * in the returned loginResult object.
         */
        binding = (SoapBindingStub) new SforceServiceLocator().getSoap();

        // Time out after a minute
        binding.setTimeout(60000);
        // Log in using the Enterprise WSDL binding
        LoginResult loginResult;
        try {
            System.out.println("LOGGING IN NOW....");
            loginResult = binding.login(userName, password);
        }
        catch (LoginFault ex) {

            // The LoginFault derives from AxisFault
            ExceptionCode exCode = ex.getExceptionCode();
            if (exCode == ExceptionCode.FUNCTIONALITY_NOT_ENABLED ||
                exCode == ExceptionCode.INVALID_CLIENT ||
                exCode == ExceptionCode.INVALID_LOGIN ||
                exCode == ExceptionCode.LOGIN_DURING_RESTRICTED_DOMAIN ||
                exCode == ExceptionCode.LOGIN_DURING_RESTRICTED_TIME ||
                exCode == ExceptionCode.ORG_LOCKED ||
                exCode == ExceptionCode.PASSWORD_LOCKOUT ||
                exCode == ExceptionCode.SERVER_UNAVAILABLE ||
                exCode == ExceptionCode.TRIAL_EXPIRED ||
                exCode == ExceptionCode.UNSUPPORTED_CLIENT) {
                System.out.println("Please be sure that you have a valid username " +
                        "and password.");
            } else {
                // Write the fault code to the console
                System.out.println(ex.getExceptionCode());
                // Write the fault message to the console
                System.out.println("An unexpected error has occurred." + ex.getMessage());
            }
            return false;
        } catch (Exception ex) {
            System.out.println("An unexpected error has occurred: " + ex.getMessage());
            ex.printStackTrace();
            return false;
        }
        // Check if the password has expired
        if (loginResult.isPasswordExpired()) {
            System.out.println("An error has occurred. Your password has expired.");
            return false;
        }

        /** Once the client application has logged in successfully, we use
         *  the results of the login call to reset the endpoint of the service
         *  to the virtual server instance that is servicing your organization.
         *  To do this, the client application sets the ENDPOINT_ADDRESS_PROPERTY
         *  of the binding object using the URL returned from the LoginResult. We
         *  use the metadata binding from this point forward as we are invoking
         *  calls in the metadata WSDL.
         */
        metadatabinding = (MetadataBindingStub)
                new MetadataServiceLocator().getMetadata();
        metadatabinding._setProperty(MetadataBindingStub.ENDPOINT_ADDRESS_PROPERTY,
                loginResult.getMetadataServerUrl());

        /** The sample client application now has an instance of the MetadataBindingStub
         *  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 the SoapBindingStub) that contains the valid sessionId
         *  for our login credentials. To do this, the sample client application
         *  creates a new SessionHeader object and set its sessionId property to the
         *  sessionId property from the LoginResult object.
         */
        // Create a new session header object and add the session id
        // from the login return object
        SessionHeader sh = new SessionHeader();
        sh.setSessionId(loginResult.getSessionId());
        /** Next, the sample client application calls the setHeader method of the
```

```
         *  SoapBindingStub to add the header to all subsequent method calls. This
         *  header will persist until the binding is destroyed or until the header
         *  is explicitly removed. The "SessionHeader" parameter is the name of the
         *  header to be added.
         */
        // set the session header for subsequent call authentication
        metadatabinding.setHeader(
            new MetadataServiceLocator().getServiceName().getNamespaceURI(),
                "SessionHeader", sh);

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

    /**
     * Create a custom object. This method demonstrates usage of the
     * create() and checkStatus() calls.
     */
    private void createCustomObject() {
        CustomObject co = new CustomObject();
        String name = "My Custom Object";
        co.setFullName("MyCustomObject" + "__c");
        co.setDeploymentStatus(DeploymentStatus.Deployed);
        co.setDescription("Created by the Metadata API Sample");
        co.setEnableActivities(true);
        co.setLabel(name);
        co.setPluralLabel(co.getLabel() + "s");
        co.setSharingModel(SharingModel.ReadWrite);
        CustomField nf = new CustomField();
        nf.setType(FieldType.Text);
        nf.setDescription("The custom object identifier on page layouts, " +
                "related lists etc");
        nf.setLabel("My Custom Object");
        nf.setFullName("MyCustomObject" + " __c");
        // The name field appears in page layouts, related lists, and elsewhere.
        co.setNameField(nf);

        try {
            AsyncResult[] ars = metadatabinding.create(new CustomObject[] { co });
            if (ars == null) {
                System.out.println("The object was not created successfully");
                return;
            }

            String createdObjectId = ars[0].getId();
            String[] ids = new String[] {createdObjectId};
            boolean done = false;
            long waitTimeMilliSecs = ONE_SECOND;
            AsyncResult[] arsStatus = null;

            /**
             * After the create() call completes, we must poll the results
             * of the checkStatus() call until it indicates that the create
             * operation is completed.
             */
            while (!done) {
                arsStatus = metadatabinding.checkStatus(ids);
                if (arsStatus == null) {
                    System.out.println("The object status cannot be retrieved");
                    return;
                }
                done = arsStatus[0].isDone();
                if (arsStatus[0].getStatusCode() != null )  {
                    System.out.println("Error status code: " +
                            arsStatus[0].getStatusCode());
                    System.out.println("Error message: " + arsStatus[0].getMessage());
                }
                Thread.sleep(waitTimeMilliSecs);
                // double the wait time for the next iteration
                waitTimeMilliSecs *= 2;
                System.out.println("The object state is " + arsStatus[0].getState());
            }

            System.out.println("The ID for the created object is " +
                    arsStatus[0].getId());
        }
        catch (Exception ex) {
            System.out.println("\nFailed to create object, error message was: \n"
                 + ex.getMessage());
            getUserInput("\nHit return to continue...");
        }

    }

    private void run() throws ServiceException {
        if (login()) {
            getUserInput("SUCCESSFUL LOGIN! Hit the enter key to continue.");
            createCustomObject();
        }
    }
}
```

USING THE METADATA API

Chapter 3

Deploying and Retrieving Metadata

Use the `deploy()` and `retrieve()` calls to move metadata (XML files) between a Salesforce.com organization and a local file system. Once you retrieve your XML files into a file system, you can manage changes in a source-code control system, copy and paste code or setup configurations, diff changes to components, and perform many other file-based development operations. At any time you can deploy those changes to another Salesforce.com organization.

Note: The Force.com IDE and Force.com Migration Tool use the `deploy()` and `retrieve()` calls to move metadata. If you use either of these tools, interaction with the Metadata API is seamless and invisible. Therefore, most developers will find it much easier to use these tools than write code that calls `deploy()` and `retrieve()` directly.

Data in XML files is formatted using the English (United States) locale. This ensures that fields that depend on locale, such as date fields, are interpreted consistently during data migrations between organizations using different languages. Organizations can support multiple languages for presentation to their users.

The `deploy()` and `retrieve()` calls are used primarily for the following development scenarios:

- Development of a custom application (or customization) in a sandbox organization. After development and testing is completed, the application or customization is then deployed into a production organization using the Metadata API.
- Team development of an application in a Developer Edition organization. After development and testing is completed, you can then distribute the application via Force.com AppExchange.

Working with the Zip File

The `deploy()` and `retrieve()` calls are used to deploy and retrieve a `.zip` file. Within the `.zip` file is a project manifest `(package.xml)` that lists what to retrieve or deploy, and one or more XML components organized into folders.

Note: A component is an instance of a metadata type. For example, `CustomObject` is a metadata type for custom objects, and the `MyCustomObject__c` component is an instance of a custom object.

The files retrieved or deployed in a `.zip` file may be unpackaged components that reside in your organization (such as *standard objects*), or packaged components that reside within named packages.

Note: The Metadata API can deploy up to 50 MB, and retrieve up to 2500 files or 400 MB at one time. If you are working with a large number of components, you should use the `listMetadata()` call to identify the subset of files that you want to retrieve or deploy. Once you know how many components you have, and of what type, you can retrieve or deploy batches of components in different `.zip` files.

Every `.zip` file contains a project manifest, a file named `package.xml`, and a set of directories that contain the components. The manifest file defines the components you are trying to retrieve or deploy in the `.zip` file.

The following is a sample `package.xml` file. Note that you can retrieve an individual component for a metadata type by specifying its `fullName` field value in a `members` element, or you can also retrieve all components of a metadata type, by using`<members>*</members>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>MyCustomObject__c</members>
        <name>CustomObject</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomTab</name>
    </types>
    <types>
        <members>Standard</members>
        <name>Profile</name>
    </types>
    <version>20.0</version>
</Package>
```

The following elements may be defined in `package.xml`:

- `<fullName>` contains the name of the server-side package. If no `<fullName>` exists, this is a client-side `unpackaged` package.
- `<types>` contains the name of the metadata type (for example, `CustomObject`) and the named members (for example, `myCustomObject__c`), to be retrieved or deployed. There can be multiple `<types>` elements in a manifest file and there is one entry for each named component, and one entry for each individual member.
- `<members>` contains the `fullName` of the component, for example `MyCustomObject__c`. The `listMetadata()` call is useful to find out the `fullName` for components of a particular metadata type, if you want to retrieve an individual component. For many metadata types, you can replace the value in `members` with the wildcard character `*` (asterisk) instead of listing each member separately. Any metadata type that has a value of **yes** in the `*` column in the Metadata Types table supports use of this wildcard.
- `<name>` contains the metadata type, for example `CustomObject` or `Profile`. There is one name defined for each metadata type in the directory. Any metadata type that extends Metadata is a valid value. The name entered must match a metadata type defined in the Metadata API WSDL. See Metadata Types for a list.
- `<version>` is the API version number used when deploying or retrieving the `.zip` file. Currently the valid value is `20.0`.

For more sample `package.xml` manifest files that show you how to work with different subsets of metadata, see Sample `package.xml` Manifest Files on page 29.

To delete items, use the same procedure but name the *manifest file* `destructiveChanges.xml` instead of `package.xml`. If you try to delete items that do not exist in an organization, the rest of the deletion will be attempted.

Metadata Types

The following table lists all of the metadata types that can be retrieved or deployed with the Metadata API, the XML name used in the `package.xml` file for the metadata type, the folder the component is retrieved into, whether or not the component may be retrieved with the wildcard (*) symbol in `package.xml`, and notes about this component, where applicable.:

Table 1: Available Metadata Types

Component	XML <name> Metadata Type	Folder	*	Notes
Action Override	`ActionOverride`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Analytic Snapshot	`AnalyticSnapshot`	`analyticsnapshots`	no	
Apex class	`ApexClass`	`classes`	yes	
Article Type	`ArticleType`	`objects`	yes	
Visualforce component	`ApexComponent`	`components`	yes	
Visualforce page	`ApexPage`	`pages`	yes	
Apex trigger	`ApexTrigger`	`triggers`	yes	
Business process	`BusinessProcess`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Custom application	`CustomApplication`	`applications`	yes	
Custom field	`CustomField`	`objects`	no	Custom fields are retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name. Individual custom fields cannot be retrieved with the wildcard (*) symbol, but must be explicitly named in `package.xml`, unless their object is named in the `CustomObject` section.
Custom label	`CustomLabels`	`labels`	yes	Custom labels that can be localized for use in different languages, countries, and currencies.
Custom object or standard object	`CustomObject`	`objects`	yes	Standard objects cannot be retrieved with the wildcard (*) symbol, but must be explicitly named in `package.xml`, and only custom fields and standard picklist fields are included. .
Custom object translation	`CustomObjectTranslation`	`objectTranslations`	yes	
Custom page web link	`CustomPageWebLink`	`weblinks`	yes	Web links are defined in a home page component.
Custom site	`CustomSite`	`sites`	yes	
Custom tab	`CustomTab`	`tabs`	yes	
Dashboard	`Dashboard`	`dashboards`	no	
Data Categories	`DataCategoryGroup`	`datacategorygroups`	yes	Includes a category group and its categories.
Document	`Document`	`document`	no	
Email template	`EmailTemplate`	`email`	no	

Component	XML <name> Metadata Type	Folder	*	Notes
Entitlement Template	`EntitlementTemplate`	`entitlementTemplates`	yes	
Home page component	`HomePageComponent`	`homePageComponents`	yes	
Home page layout	`HomePageLayout`	`homePageLayouts`	yes	
Page layout	`Layout`	`layouts`	yes	
Letterhead	`Letterhead`	`letterhead`	no	
List view	`ListView`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Lookup filter	`NamedFilter`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Picklist	`Picklist`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Portal	`Portal`	`portals`	yes	
Profile	`Profile`	`profiles`	yes	The contents of a profile retrieved depends on the contents of the organization. For example, profiles will only include field-level security for fields included in custom objects returned at the same time as the profiles.
Record type	`RecordType`	`objects`	yes	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Remote site setting	`RemoteSiteSetting`	`remoteSiteSettings`	yes	
Report	`Report`	`reports`	no	
Report type	`ReportType`	`reportTypes`	yes	Custom report types allow you to build a framework from which users can create and customize reports.
Scontrol	`Scontrol`	`scontrols`	yes	
Search layouts	`SearchLayouts`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Sharing reason	`SharingReason`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Sharing recalculation	`SharingRecalculation`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.

Component	XML <name> Metadata Type	Folder	*	Notes
Static resource	`StaticResource`	`staticResources`	yes	
Translation Workbench	`Translations`	`translations`	yes	
Validation rule	`ValidationRule`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Web link	`Weblink`	`objects`	no	This type is retrieved or deployed as part of a custom object file. You must dot-qualify the object name before the component name.
Workflow	`Workflow`	`workflows`	yes	A `.workflow` file is a container for the individual workflow components associated with an object, including `WorkflowAlert`, `WorkflowFieldUpdate`, `WorkflowOutboundMessage`,`WorkflowRule`, and `WorkflowTask`.

Sample `package.xml` Manifest Files

This section includes sample `package.xml` manifest files that show you how to work with different subsets of metadata. A manifest file can include multiple `<types>` elements so you could combine the individual samples into one `package.xml` manifest file if you want to work with all the metadata in one batch. For more information about the structure of a manifest file, see Working with the Zip File on page 25. The following samples are listed:

- Standard Objects
- All Custom Objects
- Standard Picklist Fields
- Custom Fields
- List Views for Standard Objects
- Packages

Standard Objects

This sample `package.xml` manifest file illustrates how to work with the standard Account object. Retrieving or deploying a standard object includes all custom fields and all standard picklist fields, such as the `Industry` field in Account.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>Account</members>
        <name>CustomObject</name>
    </types>
    <version>20.0</version>
</Package>
```

Note how you work with the standard Account object by specifying it as a member of a CustomObject type. However, you cannot use an asterisk wildcard to work with all standard objects; each standard object must be specified by name.

All Custom Objects

This sample `package.xml` manifest file illustrates how to work with all custom objects.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>CustomObject</name>
    </types>
    <version>20.0</version>
</Package>
```

This manifest file can be used to retrieve or deploy all custom objects. This does not include all standard objects.

Standard Picklist Fields

This sample `package.xml` manifest file illustrates how to work with the standard `Industry` picklist field in the standard Account object.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>Account.Industry</members>
        <name>CustomField</name>
    </types>
    <version>20.0</version>
</Package>
```

Note the ***`objectName.picklistField`*** syntax in the `<members>` field where *`objectName`* is the name of the object, such as `Account`, and *`picklistField`* is the name of the standard picklist field, such as `Industry`.

Custom Fields

This sample `package.xml` manifest file illustrates how to work with custom fields in custom and standard objects.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>MyCustomObject__c.MyCustomField__c</members>
        <name>CustomField</name>
    </types>
    <types>
        <members>Account.SLA__c</members>
        <name>CustomField</name>
    </types>
    <version>20.0</version>
</Package>
```

Note the ***`objectName.customField`*** syntax in the `<members>` field where *`objectName`* is the name of the object, such as Account, and *`customField`* is the name of the custom field, such as an `SLA` picklist field representing a service-level agreement option. The `MyCustomField` custom field in the MyCustomObject custom object is uniquely identified by its full name, `MyCustomObject__c.MyCustomField__c`.

List Views for Standard Objects

The easiest way to retrieve list views for a standard object is to retrieve the object. The list views are included in the retrieved component. See Standard Objects on page 29.

You can also work with individual list views if you do not want to retrieve all the details for the object. This sample `package.xml` manifest file illustrates how to work with a list view for the standard Account object.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>Account.AccountTeam</members>
        <name>ListView</name>
    </types>
    <version>20.0</version>
</Package>
```

Note the ***`objectName.listViewUniqueName`*** syntax in the `<members>` field where *`objectName`* is the name of the object, such as Account, and *`listViewUniqueName`* is the `View Unique Name` for the list view. If you retrieve this list view, the component is stored in `objects/Account.object`.

Packages

To retrieve a package, set the name of the package in the `packageNames` field in RetrieveRequest when you call `retrieve()`. The `package.xml` manifest file is automatically populated in the retrieved `.zip` file. The `<fullName>` element in `package.xml` contains the name of the retrieved package.

If you use an asterisk wildcard in a `<members>` element to retrieve all the components of a particular metadata type, the retrieved contents do not include components in managed packages. For more information about managed packages, see the Force.com Quick Reference for Developing Packages.

The easiest way to retrieve a component in a managed package is to retrieve the complete package by setting the name of the package in the `packageNames` field in RetrieveRequest, as described above. The following sample `package.xml` manifest file illustrates an alternative to retrieve an individual component in a package.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>myns__MyCustomObject__c</members>
        <name>CustomObject</name>
    </types>
    <version>20.0</version>
</Package>
```

Note the ***`namespacePrefix__`****`objectName`* syntax in the `<members>` field where *`namespacePrefix`* is the namespace prefix of the package and *`objectName`* is the name of the object. A namespace prefix is a one to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other publishers. For more information about namespace prefixes, see “Registering a Namespace Prefix” in the Salesforce.com online help.

Running Tests in a Deployment

For deployment to a production organization, all the tests in your organization, except for those that originate from installed managed packages, are automatically run. If any of the tests fail, the entire deployment will roll back.

There is an exception to this rule if you are deploying components for one or more of the following metadata types:

- ApexComponent
- ApexPage
- Dashboard
- EmailTemplate

- Report
- Scontrol
- StaticResource

If your deployment consists entirely of components for one or more of these metadata types, no tests are run. However, if the deployment includes components for any other metadata type, all the tests are automatically run.

For example, no tests are run for the following deployments:

- One ApexComponent component
- 100 Report components and 40 Dashboard components

All tests are automatically run for the following deployments:

- One CustomField component
- One ApexComponent component and one ApexClass component
- Five CustomField components and one ApexPage component
- 100 Report components, 40 Dashboard components, and one CustomField component

See Also:

deploy()

Chapter 4

CRUD-Based Metadata Development

Use the CRUD-based metadata calls to create, update, or delete setup and configuration components for your organization or application. These configuration components include custom objects, custom fields, and other configuration metadata. The metadata calls mimic the behavior in the Salesforce.com user interface for creating, updating, or deleting components. Whatever rules apply there also apply to these calls.

Note: CRUD (create, read, update, delete) implies that there is a read call, but there is no equivalent read call for CRUD-based development. If you want to read your metadata, you should use the `retrieve()` call, described in File-Based Development on page 5.

Metadata calls are different from the core, synchronous API calls in the following ways:

- Metadata API calls are available in a separate WSDL. To download the WSDL, log into Salesforce.com, select ***Your Name*** ➤ **Setup** ➤ **Develop** ➤ **API** and click the **Download Metadata WSDL** link.
- After logging in, you must send Metadata API calls to the Metadata API endpoint, which has a different URL than the Web services API. Retrieve the `metadataServerUrl` from the LoginResult returned by your Web services API `login()` call. For more information about the Web services API, see the Web Services API Developer's Guide.
- There are three metadata calls with the same name as the corresponding core synchronous calls but with different signatures: `create()`, `update()`, and `delete()`. There is also a special utility call, `checkStatus()`, which you use to poll for the completion of the asynchronous call.
- Metadata calls are asynchronous, which means that the results are not returned in a single call; the API core calls are synchronous; the results are returned in one call.
- The responses returned are all of type AsyncResult, unlike core API calls, which return different result types.

The following development workflow is common for CRUD-based metadata calls:

1. The *logged-in user* issues a metadata call, specifying all required fields to be created or updated.
2. Salesforce.com returns an AsyncResult object for each component you tried to create or update. The AsyncResult object is updated with status information as the operation moves from a queue to completed or error state.
3. Call `checkStatus()` in a loop until the status values in AsyncResult indicate that all the create or update operations are completed. Start with a wait time of one second between iterations of `checkStatus()` calls, and double the wait time each time you make a subsequent call.

Note: There are two metadata calls that support retrieving and deploying metadata components. For more information, see Deploying and Retrieving Metadata.

REFERENCE

Chapter 5

File-Based Calls

Use the following file-based calls to deploy or retrieve XML components.

- `deploy()`
- `retrieve()`

`deploy()`

Uses file representations of components to create, update, or delete those components in an organization.

Syntax

```
AsyncResult = metadatabinding.deploy(base64 zipFile, DeployOptions deployOptions)
```

Usage

Use this call to take file representations of components and deploy them into an organization by creating, updating, or deleting the components they represent.

Note: The Metadata API can deploy up to 50 MB, and retrieve up to 2500 files or 400 MB at one time. If you are working with a large number of components, you should use the `listMetadata()` call to identify the subset of files that you want to deploy, or you should deploy batches of components in different `.zip` files.

To deploy (create or update) packaged or unpackaged components:

1. Issue a `deploy()` call to start the asynchronous deployment. An AsyncResult object is returned. If the call is completed, the `done` field contains `true`. Most often, the call is not completed quickly enough to be noted in the first result. If it is completed, note the value in the `id` field returned and skip the next step.
2. If the call is not complete, issue a `checkStatus()` call in a loop using the value in the `id` field of the AsyncResult object returned by the `deploy()` call in the previous step. Check the AsyncResult object which is returned until the `done` field contains `true`. The time taken to complete a `deploy()` call depends on the size of the zip file being deployed, so a longer wait time between iterations should be used as the size of the zip file increases.
3. Issue a `checkDeployStatus()` call to obtain the results of the `deploy()` call, using the `id` value returned in the first step.

To delete items, use the same procedure but name the *manifest file* `destructiveChanges.xml` instead of `package.xml`. If you try to delete items that do not exist in an organization, the rest of the deletion will be attempted.

To track the status of deployments that are in progress or completed in the last 24 hours, click ***Your Name*** ➤ **Setup** ➤ **Deploy** ➤ **Monitor Deployments**.

Permissions

Your client application must be logged in with the "Modify All Data" permission.

Sample Code—Java

This sample shows how to deploy components in a zip file. See the `retrieve()` sample code for details on how to retrieve a zip file.

```
package com.doc.samples;

import java.io.*;

import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;

import com.sforce.soap._2006._04.metadata.AsyncRequestState;
import com.sforce.soap._2006._04.metadata.AsyncResult;
import com.sforce.soap._2006._04.metadata.MetadataBindingStub;
import com.sforce.soap._2006._04.metadata.MetadataServiceLocator;
import com.sforce.soap._2006._04.metadata.DeployOptions;
import com.sforce.soap._2006._04.metadata.DeployResult;
import com.sforce.soap._2006._04.metadata.DeployMessage;
import com.sforce.soap._2006._04.metadata.RunTestsResult;
import com.sforce.soap._2006._04.metadata.RunTestFailure;
import com.sforce.soap._2006._04.metadata.CodeCoverageWarning;
import com.sforce.soap.enterprise.LoginResult;
import com.sforce.soap.enterprise.SessionHeader;
import com.sforce.soap.enterprise.SforceServiceLocator;
import com.sforce.soap.enterprise.SoapBindingStub;
import com.sforce.soap.enterprise.fault.ExceptionCode;
import com.sforce.soap.enterprise.fault.LoginFault;

/**
 * Deploy a zip file of metadata components.
 * Prerequisite: Have a deploy.zip file that includes a package.xml manifest file that
 * details the contents of the zip file.
 */
public class DeploySample {
    // binding for the Enterprise WSDL used for login() call
    private SoapBindingStub binding;
    // binding for the metadata WSDL used for create() and checkStatus() calls
    private MetadataBindingStub metadatabinding;

    static BufferedReader rdr = new BufferedReader(new InputStreamReader(System.in));

    private static final String ZIP_FILE = "deploy.zip";

    // one second in milliseconds
    private static final long ONE_SECOND = 1000;
    // maximum number of attempts to deploy the zip file
    private static final int MAX_NUM_POLL_REQUESTS = 50;

    public static void main(String[] args) throws ServiceException, Exception {
        DeploySample sample = new DeploySample();
        sample.run();
    }

    private void run() throws ServiceException, Exception {
        if (login()) {
            getUserInput("SUCCESSFUL LOGIN! Hit the enter key to continue.");
            deployZip();
        }
    }

    private void deployZip()
        throws RemoteException, Exception
    {
        byte zipBytes[] = readZipFile();
        DeployOptions deployOptions = new DeployOptions();
        deployOptions.setPerformRetrieve(false);
        deployOptions.setRollbackOnError(true);
        AsyncResult asyncResult = metadatabinding.deploy(zipBytes, deployOptions);

        // Wait for the deploy to complete
        int poll = 0;
        long waitTimeMilliSecs = ONE_SECOND;
        while (!asyncResult.isDone()) {
            Thread.sleep(waitTimeMilliSecs);
            // double the wait time for the next iteration
            waitTimeMilliSecs *= 2;
            if (poll++ > MAX_NUM_POLL_REQUESTS) {
                throw new Exception("Request timed out. If this is a large set " +
                        "of metadata components, check that the time allowed by " +
                        "MAX_NUM_POLL_REQUESTS is sufficient.");
            }
            asyncResult = metadatabinding.checkStatus(
                    new String[] {asyncResult.getId()})[0];
            System.out.println("Status is: " + asyncResult.getState());
        }

        if (asyncResult.getState() != AsyncRequestState.Completed) {
            throw new Exception(asyncResult.getStatusCode() + " msg: " +
                    asyncResult.getMessage());
        }

        DeployResult result = metadatabinding.checkDeployStatus(asyncResult.getId());
        if (!result.isSuccess()) {
            printErrors(result);
            throw new Exception("The files were not successfully deployed");
        }

        System.out.println("The file " + ZIP_FILE + " was successfully deployed");
    }

    /**
     * Read in the zip file contents into a byte array.
     * @return byte[]
     * @throws Exception - if cannot find the zip file to deploy
     */
    private byte[] readZipFile()
        throws Exception
    {
        // We assume here that you have a deploy.zip file.
        // See the retrieve sample for how to retrieve a zip file.
        File deployZip = new File(ZIP_FILE);
        if (!deployZip.exists() || !deployZip.isFile())
            throw new Exception("Cannot find the zip file to deploy. Looking for " +
                    deployZip.getAbsolutePath());

        FileInputStream fos = new FileInputStream(deployZip);
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        int readbyte = -1;
        while ((readbyte = fos.read()) != -1)  {
            bos.write(readbyte);
        }
        fos.close();
        bos.close();
        return bos.toByteArray();
    }


    /**
     * Print out any errors, if any, related to the deploy.
     * @param result - DeployResult
     */
    private void printErrors(DeployResult result)
    {
        DeployMessage messages[] = result.getMessages();
        StringBuilder buf = new StringBuilder("Failures:\n");
        for (DeployMessage message : messages) {
            if (!message.isSuccess()) {
                String loc = (message.getLineNumber() == null ? "" :
                    ("(" + message.getLineNumber() + "," +
                            message.getColumnNumber() + ")"));
                if (loc.length() == 0
                        && !message.getFileName().equals(message.getFullName())) {
                    loc = "(" + message.getFullName() + ")";
                }
                buf.append(message.getFileName() + loc + ":" +
                        message.getProblem()).append('\n');
            }
        }
        RunTestsResult rtr = result.getRunTestResult();
        if (rtr.getFailures() != null) {
            for (RunTestFailure failure : rtr.getFailures()) {
                String n = (failure.getNamespace() == null ? "" :
                    (failure.getNamespace() + ".")) + failure.getName();
                buf.append("Test failure, method: " + n + "." +
                        failure.getMethodName() + " -- " +
                        failure.getMessage() + " stack " +
                        failure.getStackTrace() + "\n\n");
            }
        }
        if (rtr.getCodeCoverageWarnings() != null) {
            for (CodeCoverageWarning ccw : rtr.getCodeCoverageWarnings()) {
                buf.append("Code coverage issue");
                if (ccw.getName() != null) {
                    String n = (ccw.getNamespace() == null ? "" :
                        (ccw.getNamespace() + ".")) + ccw.getName();
                    buf.append(", class: " + n);
                }
                buf.append(" -- " + ccw.getMessage() + "\n");
            }
        }

        System.out.println(buf.toString());
    }

    /**
     * The login call is used to obtain a token from Salesforce.
     * This token must be passed to all other calls to provide
     * authentication.
     */
    private boolean login() throws ServiceException {
        String userName = getUserInput("Enter username: ");
        String password = getUserInput("Enter password: ");
        /** Next, the sample client application initializes the binding stub.
         *
         * This is our main interface to the API for the Enterprise WSDL.
         * The getSoap method takes an optional parameter,
         * (a java.net.URL) which is the endpoint.
         * For the login call, the parameter always starts with
         * http(s)://login.salesforce.com. After logging in, the sample
         * client application changes the endpoint to the one specified
         * in the returned loginResult object.
         */
        binding = (SoapBindingStub) new SforceServiceLocator().getSoap();

        // Time out after a minute
        binding.setTimeout(60000);
        // Log in using the Enterprise WSDL binding
        LoginResult loginResult;
        try {
            System.out.println("LOGGING IN NOW....");
            loginResult = binding.login(userName, password);
        }
        catch (LoginFault ex) {

            // The LoginFault derives from AxisFault
            ExceptionCode exCode = ex.getExceptionCode();
            if (exCode == ExceptionCode.FUNCTIONALITY_NOT_ENABLED ||
                exCode == ExceptionCode.INVALID_CLIENT ||
                exCode == ExceptionCode.INVALID_LOGIN ||
                exCode == ExceptionCode.LOGIN_DURING_RESTRICTED_DOMAIN ||
                exCode == ExceptionCode.LOGIN_DURING_RESTRICTED_TIME ||
                exCode == ExceptionCode.ORG_LOCKED ||
                exCode == ExceptionCode.PASSWORD_LOCKOUT ||
                exCode == ExceptionCode.SERVER_UNAVAILABLE ||
                exCode == ExceptionCode.TRIAL_EXPIRED ||
                exCode == ExceptionCode.UNSUPPORTED_CLIENT) {
                System.out.println("Please be sure that you have a valid username " +
                        "and password.");
            } else {
                // Write the fault code to the console
                System.out.println(ex.getExceptionCode());
                // Write the fault message to the console
                System.out.println("An unexpected error has occurred." + ex.getMessage());
            }
            return false;
        } catch (Exception ex) {
            System.out.println("An unexpected error has occurred: " + ex.getMessage());
            ex.printStackTrace();
            return false;
        }
        // Check if the password has expired
        if (loginResult.isPasswordExpired()) {
            System.out.println("An error has occurred. Your password has expired.");
            return false;
        }

        /** Once the client application has logged in successfully, we use
         *  the results of the login call to reset the endpoint of the service
         *  to the virtual server instance that is servicing your organization.
         *  To do this, the client application sets the ENDPOINT_ADDRESS_PROPERTY
         *  of the binding object using the URL returned from the LoginResult. We
         *  use the metadata binding from this point forward as we are invoking
         *  calls in the metadata WSDL.
         */
        metadatabinding = (MetadataBindingStub)
                new MetadataServiceLocator().getMetadata();
        metadatabinding._setProperty(MetadataBindingStub.ENDPOINT_ADDRESS_PROPERTY,
                loginResult.getMetadataServerUrl());

        /** The sample client application now has an instance of the MetadataBindingStub
         *  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 the SoapBindingStub) that contains the valid sessionId
         *  for our login credentials. To do this, the sample client application
         *  creates a new SessionHeader object and set its sessionId property to the
         *  sessionId property from the LoginResult object.
         */
        // Create a new session header object and add the session id
        // from the login return object
        SessionHeader sh = new SessionHeader();
        sh.setSessionId(loginResult.getSessionId());
        /** Next, the sample client application calls the setHeader method of the
         *  SoapBindingStub to add the header to all subsequent method calls. This
         *  header will persist until the binding is destroyed or until the header
         *  is explicitly removed. The "SessionHeader" parameter is the name of the
         *  header to be added.
         */
        // set the session header for subsequent call authentication
        metadatabinding.setHeader(
            new MetadataServiceLocator().getServiceName().getNamespaceURI(),
                "SessionHeader", sh);

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

    //The sample client application retrieves the user's login credentials.
    // Helper function for retrieving user input from the console
    String getUserInput(String prompt) {
        System.out.print(prompt);
        try {
            return rdr.readLine();
        }
        catch (IOException ex) {
            return null;
        }
    }

}
```

Arguments

Name	Type	Description
`zipFile`	base64	Base 64-encoded binary data. Client applications must encode the binary data as base64.
`deployOptions`	DeployOptions	Encapsulates options for determining which packages or files are deployed.

DeployOptions

The following deployment options can be selected for this call:

Name	Type	Description
`allowMissingFiles`	boolean	Specifies whether a deploy succeeds even if files that are specified in `package.xml` but are not in the `.zip` file (`true` or not `false`). Do not set this argument for deployment to *production organizations*.
`autoUpdatePackage`	boolean	If a file is in the `.zip` file but not specified in the `package.xml`, specifies whether the file should be automatically added to the package (`true` or not `false`).

Name	Type	Description
		A `retrieve()` is automatically issued with the updated `package.xml` that includes the `.zip` file. Do not set this argument for deployment to *production organizations*.
`checkOnly`	boolean	Indicates whether Apex classes and triggers are saved to the organization as part of the deployment (`false`) or not (`true`). Defaults to `false`. Any errors or messages that would have been issued are still generated. This parameter is similar to the Salesforce.com Ant tool's `checkOnly` parameter.
`ignoreWarnings`	boolean	Indicates whether a warning should allow a deployment to complete successfully (`true`) or not (`false`). Defaults to `false`. The DeployMessage object for a warning contains the following values: • `problemType`—`Warning` • `problem`—The text of the warning. If a warning occurs and `ignoreWarnings` is set to `true`, the `success` field in DeployMessage is `true`. If `ignoreWarnings` is set to `false`, `success` is set to `false` and the warning is treated like an error. This field is available in API version 18.0 and later. Prior to version 18.0, there was no distinction between warnings and errors. All problems were treated as errors and prevented a successful deployment.
`performRetrieve`	boolean	Indicates whether a `retrieve()` call is performed immediately after the deployment (`true`) or not (`false`). Set to `true` in order to retrieve whatever was just deployed.
`rollbackOnError`	boolean	Indicates whether any failure causes a complete rollback (`true`) or not (`false`). If `false`, whatever set of actions can be performed without errors are performed, and errors are returned for the remaining actions. This parameter must be set to `true` if you are deploying to a production organization.
`runAllTests`	boolean	If `true`, all Apex tests defined in the organization are run. For deployment to a production organization, all tests, except for those that originate from installed managed packages, are automatically run regardless of this argument. If any of the tests fail when the `rollbackOnError` parameter is set to true, the entire deployment will roll back.
`runTests`	string[]	A list of Apex tests to be run during deployment. Specify the class name, one name per instance. The class name

Name	Type	Description

		may also specify a namespace with a dot. For example, to run three tests:
		`<runTests>positive_test</runTests>`
		`<runTests>negative_test</runTests>`
		`<runTests>namespace.third_test</runTests>`
		If any of these tests fail when the `rollbackOnError` parameter is set to `true`, the deployment is rolled back and no changes will be made to your organization.

`singlePackage`	boolean	Indicates whether the specified `.zip` file points to a directory structure with a single package (`true`) or a set of packages (`false`).

Response

AsyncResult

See Also:

Running Tests in a Deployment

checkDeployStatus()

Checks the status of declarative metadata call `deploy()`.

Syntax

```
DeployResult = metadatabinding.checkDeployStatus(ID id);
```

Usage

`checkDeployStatus` is used as part of the process for deploying packaged or unpackaged components to an organization:

1. Issue a `deploy()` call to start the asynchronous deployment. An AsyncResult object is returned. If the call is completed, the `done` field contains `true`. Most often, the call is not completed quickly enough to be noted in the first result. If it is completed, note the value in the `id` field returned and skip the next step.
2. If the call is not complete, issue a `checkStatus()` call in a loop using the value in the `id` field of the AsyncResult object returned by the `deploy()` call in the previous step. Check the AsyncResult object which is returned until the `done` field contains `true`. The time taken to complete a `deploy()` call depends on the size of the zip file being deployed, so a longer wait time between iterations should be used as the size of the zip file increases.
3. Issue a `checkDeployStatus()` call to obtain the results of the `deploy()` call, using the `id` value returned in the first step.

Sample Code—Java

See the `deploy()` sample code for sample usage of this call.

Arguments

Name	Type	Description
`id`	ID	ID obtained from an AsyncResult object returned by `deploy()` or a subsequent `checkDeployStatus()` call.

Response

DeployResult

`retrieve()`

This call retrieves XML file representations of components in an organization.

Syntax

```
AsyncResult = metadatabinding.retrieve(RetrieveRequest retrieveRequest)
```

Usage

Use this call to retrieve file representations of components in an organization.

Note: The Metadata API can deploy up to 50 MB, and retrieve up to 2500 files or 400 MB at one time. If you are working with a large number of components, you should use the `listMetadata()` call to identify the subset of files that you want to retrieve, or you should retrieve batches of components in different `.zip` files.

To retrieve packaged or unpackaged components:

1. Issue a `retrieve()` call to start the asynchronous retrieval. An AsyncResult object is returned. If the call is completed, the `done` field contains `true`. Most often, the call is not completed quickly enough to be noted in the result. If it is completed, note the value in the `id` field returned and skip the next step.
2. If the call is not complete, issue a `checkStatus()` call in a loop using the value in the `id` field of the AsyncResult object returned by the `retrieve()` call in the previous step. Check the AsyncResult object returned, until the `done` field contains `true`. The time taken to complete a `retrieve()` call depends on the size of the zip file being deployed, so a longer wait time between iterations should be used as the size of the zip file increases.
3. Issue a `checkRetrieveStatus()` call to obtain the results of the `retrieve()` call, using the `id` value returned in the first step.

Permissions

Your client application must be logged in with the “Modify All Data” permission.

Sample Code—Java

This sample shows how to retrieve components into a zip file. See the `deploy()` sample code for details on how to deploy a zip file.

Note: This sample was created using Apache Axis. The WSDL2Java utility generates a `_package` class, even though the metadata type is defined as `Package` in the Metadata WSDL. Other SOAP clients may generate a different name for the `_package` class.

```
package com.doc.samples;

import java.io.*;
import java.util.*;
import java.nio.ByteBuffer;
import java.nio.channels.Channels;
import java.nio.channels.FileChannel;
import java.nio.channels.ReadableByteChannel;
import java.nio.channels.WritableByteChannel;

import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import com.sforce.soap.enterprise.LoginResult;
import com.sforce.soap.enterprise.SessionHeader;
import com.sforce.soap.enterprise.SforceServiceLocator;
import com.sforce.soap.enterprise.SoapBindingStub;
import com.sforce.soap.enterprise.fault.ExceptionCode;
import com.sforce.soap.enterprise.fault.LoginFault;

import com.sforce.soap._2006._04.metadata.MetadataBindingStub;
import com.sforce.soap._2006._04.metadata.MetadataServiceLocator;
import com.sforce.soap._2006._04.metadata.AsyncResult;
import com.sforce.soap._2006._04.metadata.RetrieveRequest;
import com.sforce.soap._2006._04.metadata.AsyncRequestState;
import com.sforce.soap._2006._04.metadata.RetrieveResult;
import com.sforce.soap._2006._04.metadata.RetrieveMessage;
// Note that Axis generates a _package class, even though it is defined as Package
// in the WSDL. Other SOAP clients may generate a different name for the _package class.
import com.sforce.soap._2006._04.metadata._package;
import com.sforce.soap._2006._04.metadata.PackageTypeMembers;


public class RetrieveSample {
    // binding for the Enterprise WSDL used for login() call
 private SoapBindingStub binding;
    // binding for the metadata WSDL used for create() and checkStatus() calls
 private MetadataBindingStub metadatabinding;

 static BufferedReader rdr = new BufferedReader(new InputStreamReader(System.in));

    // one second in milliseconds
    private static final long ONE_SECOND = 1000;
    // maximum number of attempts to retrieve the results
    private static final int MAX_NUM_POLL_REQUESTS = 50;

    // manifest file that controls which components get retrieved
    private static final String MANIFEST_FILE = "package.xml";

    private static final double API_VERSION = 15.0;

    public static void main(String[] args) throws ServiceException, Exception {
        RetrieveSample sample = new RetrieveSample();
        sample.run();
    }

    private void run() throws ServiceException, Exception {
        if (login()) {
            getUserInput("SUCCESSFUL LOGIN! Hit the enter key to continue.");
            retrieveZip();
        }
    }


    private void retrieveZip() throws RemoteException, Exception
    {
        RetrieveRequest retrieveRequest = new RetrieveRequest();
        retrieveRequest.setApiVersion(API_VERSION);
        setUnpackaged(retrieveRequest);

        AsyncResult asyncResult = metadatabinding.retrieve(retrieveRequest);
        // Wait for the retrieve to complete
        int poll = 0;
        long waitTimeMilliSecs = ONE_SECOND;
        while (!asyncResult.isDone()) {
            Thread.sleep(waitTimeMilliSecs);
            // double the wait time for the next iteration
            waitTimeMilliSecs *= 2;
            if (poll++ > MAX_NUM_POLL_REQUESTS) {
                throw new Exception("Request timed out.  If this is a large set " +
                  "of metadata components, check that the time allowed " +
                  "by MAX_NUM_POLL_REQUESTS is sufficient.");
            }
            asyncResult = metadatabinding.checkStatus(
              new String[] {asyncResult.getId()})[0];
            System.out.println("Status is: " + asyncResult.getState());
        }

        if (asyncResult.getState() != AsyncRequestState.Completed) {
            throw new Exception(asyncResult.getStatusCode() + " msg: " +
                    asyncResult.getMessage());
        }

        RetrieveResult result = metadatabinding.checkRetrieveStatus(asyncResult.getId());

        // Print out any warning messages
        StringBuilder buf = new StringBuilder();
        if (result.getMessages() != null) {
            for (RetrieveMessage rm : result.getMessages()) {
                buf.append(rm.getFileName() + " - " + rm.getProblem());
            }
        }
        if (buf.length() > 0) {
            System.out.println("Retrieve warnings:\n" + buf);
        }

        // Write the zip to the file system
        System.out.println("Writing results to zip file");
        ByteArrayInputStream bais = new ByteArrayInputStream(result.getZipFile());
        File resultsFile = new File("retrieveResults.zip");
        FileOutputStream os = new FileOutputStream(resultsFile);
        try {
            ReadableByteChannel src = Channels.newChannel(bais);
            FileChannel dest = os.getChannel();
            copy(src, dest);

            System.out.println("Results written to " + resultsFile.getAbsolutePath());
        }
        finally {
            os.close();
        }

    }

    /**
     * Helper method to copy from a readable channel to a writable channel,
     * using an in-memory buffer.
     */
    private void copy(ReadableByteChannel src, WritableByteChannel dest)
        throws IOException
    {
        // use an in-memory byte buffer
        ByteBuffer buffer = ByteBuffer.allocate(8092);
        while (src.read(buffer) != -1) {
            buffer.flip();
            while(buffer.hasRemaining()) {
                dest.write(buffer);
            }
            buffer.clear();
        }
    }

    private void setUnpackaged(RetrieveRequest request) throws Exception
    {
        // Edit the path, if necessary, if your package.xml file is located elsewhere
        File unpackedManifest = new File(MANIFEST_FILE);
        System.out.println("Manifest file: " + unpackedManifest.getAbsolutePath());

        if (!unpackedManifest.exists() || !unpackedManifest.isFile())
            throw new Exception("Should provide a valid retrieve manifest " +
                    "for unpackaged content. " +
                    "Looking for " + unpackedManifest.getAbsolutePath());

        // Note that we populate the _package object by parsing a manifest file here.
        // You could populate the _package based on any source for your
        // particular application.
        _package p = parsePackage(unpackedManifest);
        request.setUnpackaged(p);
    }

    private _package parsePackage(File file) throws Exception {
        try {
            InputStream is = new FileInputStream(file);
            List<PackageTypeMembers> pd = new ArrayList<PackageTypeMembers>();
            DocumentBuilder db =
                DocumentBuilderFactory.newInstance().newDocumentBuilder();
            Element d = db.parse(is).getDocumentElement();
            for (Node c = d.getFirstChild(); c != null; c = c.getNextSibling()) {
                if (c instanceof Element) {
                    Element ce = (Element)c;
                    //
                    NodeList namee = ce.getElementsByTagName("name");
                    if (namee.getLength() == 0) {
                        // not
                        continue;
                    }
                    String name = namee.item(0).getTextContent();
                    NodeList m = ce.getElementsByTagName("members");
                    List<String> members = new ArrayList<String>();
                    for (int i = 0; i < m.getLength(); i++) {
                        Node mm = m.item(i);
                        members.add(mm.getTextContent());
                    }
                    PackageTypeMembers pdi = new PackageTypeMembers();
                    pdi.setName(name);
                    pdi.setMembers(members.toArray(new String[members.size()]));
                    pd.add(pdi);
                }
            }
            _package r = new _package();
            r.setTypes(pd.toArray(new PackageTypeMembers[pd.size()]));
            r.setVersion(API_VERSION + "");
            return r;
        } catch (ParserConfigurationException pce) {
            throw new Exception("Cannot create XML parser", pce);
        } catch (IOException ioe) {
            throw new Exception(ioe);
        } catch (SAXException se) {
            throw new Exception(se);
        }
    }

    /**
     * The login call is used to obtain a token from Salesforce.
     * This token must be passed to all other calls to provide
     * authentication.
     */
    private boolean login() throws ServiceException {
        String userName = getUserInput("Enter username: ");
        String password = getUserInput("Enter password: ");
        /** Next, the sample client application initializes the binding stub.
          *
          * This is our main interface to the API for the Enterprise WSDL.
          * The getSoap method takes an optional parameter,
          * (a java.net.URL) which is the endpoint.
          * For the login call, the parameter always starts with
          * http(s)://login.salesforce.com. After logging in, the sample
          * client application changes the endpoint to the one specified
          * in the returned loginResult object.
          */
        binding = (SoapBindingStub) new SforceServiceLocator().getSoap();

        // Time out after a minute
        binding.setTimeout(60000);
        // Log in using the Enterprise WSDL binding
        LoginResult loginResult;
        try {
            System.out.println("LOGGING IN NOW....");
            loginResult = binding.login(userName, password);
        }
        catch (LoginFault ex) {

            // The LoginFault derives from AxisFault
            ExceptionCode exCode = ex.getExceptionCode();
            if (exCode == ExceptionCode.FUNCTIONALITY_NOT_ENABLED ||
                exCode == ExceptionCode.INVALID_CLIENT ||
                exCode == ExceptionCode.INVALID_LOGIN ||
                exCode == ExceptionCode.LOGIN_DURING_RESTRICTED_DOMAIN ||
                exCode == ExceptionCode.LOGIN_DURING_RESTRICTED_TIME ||
                exCode == ExceptionCode.ORG_LOCKED ||
                exCode == ExceptionCode.PASSWORD_LOCKOUT ||
                exCode == ExceptionCode.SERVER_UNAVAILABLE ||
                exCode == ExceptionCode.TRIAL_EXPIRED ||
                exCode == ExceptionCode.UNSUPPORTED_CLIENT) {
                System.out.println("Please be sure that you have a valid username " +
                        "and password.");
            } else {
                // Write the fault code to the console
                System.out.println(ex.getExceptionCode());
                // Write the fault message to the console
                System.out.println("An unexpected error has occurred." + ex.getMessage());
            }
            return false;
        } catch (Exception ex) {
            System.out.println("An unexpected error has occurred: " + ex.getMessage());
            ex.printStackTrace();
            return false;
        }
        // Check if the password has expired
        if (loginResult.isPasswordExpired()) {
            System.out.println("An error has occurred. Your password has expired.");
            return false;
        }

        /** Once the client application has logged in successfully, we use
         *  the results of the login call to reset the endpoint of the service
         *  to the virtual server instance that is servicing your organization.
         *  To do this, the client application sets the ENDPOINT_ADDRESS_PROPERTY
         *  of the binding object using the URL returned from the LoginResult. We
         *  use the metadata binding from this point forward as we are invoking
         *  calls in the metadata WSDL.
         */
        metadatabinding = (MetadataBindingStub)
                new MetadataServiceLocator().getMetadata();
        metadatabinding._setProperty(MetadataBindingStub.ENDPOINT_ADDRESS_PROPERTY,
                loginResult.getMetadataServerUrl());

        /** The sample client application now has an instance of the MetadataBindingStub
         *  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 the SoapBindingStub) that contains the valid sessionId
         *  for our login credentials. To do this, the sample client application
         *  creates a new SessionHeader object and set its sessionId property to the
         *  sessionId property from the LoginResult object.
         */
        // Create a new session header object and add the session id
        // from the login return object
        SessionHeader sh = new SessionHeader();
        sh.setSessionId(loginResult.getSessionId());
        /** Next, the sample client application calls the setHeader method of the
         *  SoapBindingStub to add the header to all subsequent method calls. This
         *  header will persist until the binding is destroyed or until the header
         *  is explicitly removed. The "SessionHeader" parameter is the name of the
         *  header to be added.
         */
        // set the session header for subsequent call authentication
        metadatabinding.setHeader(
            new MetadataServiceLocator().getServiceName().getNamespaceURI(),
                "SessionHeader", sh);

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

    //The sample client application retrieves the user's login credentials.
    // Helper function for retrieving user input from the console
    String getUserInput(String prompt) {
        System.out.print(prompt);
        try {
            return rdr.readLine();
        }
        catch (IOException ex) {
            return null;
        }
    }

}
```

Arguments

Name	Type	Description
`retrieveRequest`	RetrieveRequest	Encapsulates options for determining which packages or files are retrieved.

Response

AsyncResult

RetrieveRequest

The RetrieveRequest object specified in a `retrieve()` call consists of the following properties:

Name	Type	Description
`apiVersion`	double	Required. The API version for the retrieve request. The API version determines the fields retrieved for each metadata type. For example, an `icon` field was added to the `CustomTab` for API version 14.0. If you retrieve components for version 13.0 or earlier, the components will not include the `icon` field.
`packageNames`	string[]	A list of package names to be retrieved. If you are retrieving only unpackaged components, do not specify a name here. You can retrieve packaged and unpackaged components in the same retrieve.
`singlePackage`	boolean	Specifies whether only a single package is being retrieved (`true`) or not (`false`). If `false`, then more than one package is being retrieved.
`specificFiles`	string[]	A list of file names to be retrieved. If a value is specified for this property, `packageNames` must be set to `null` and `singlePackage` must be set to `true`.
`unpackaged`	Package	A list of components to retrieve that are not in a package.

Package

This extension of Metadata is specified in a RetrieveRequest as part of a `retrieve()` call. Use it to specify metadata components to be retrieved.

Name	Type	Description
`apiAccessLevel`	APIAccessLevel (enumeration of type string)	Package components have access via dynamic Apex and the API to standard and custom objects in the organization where they are installed. Administrators who install packages may wish to restrict this access after installation for improved security. The valid values are: • Unrestricted—Package components have the same API access to standard objects as the user who is logged in when the component sends a request to the API.

Name	Type	Description
		• Restricted—The administrator can select which standard objects the components can access. Further, the components in restricted packages can only access custom objects in the current package if the user profile provides access to them. For more information, see "About API and Dynamic Apex Access in Packages" in the Salesforce.com online help.
`description`	string	A short description of the package.
`fullName`	string	The package developer name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`namespacePrefix`	string	The namespace of the developer organization where the package was created.
`objectPermissions`	ProfileObjectPermissions[]	Indicates which objects are accessible to the package, and the kind of access available (create, read, update, delete).
`setupWeblink`	string	The weblink used to describe package installation.
`types`	PackageTypeMembers[]	A PackageTypeMembers object for each type of component being retrieved.
`version`	string	Required. The version of the component type.

PackageTypeMembers

This object is specified in a Package component as part of a `retrieve()` call. Each PackageTypeMembers contains the following arguments:

Name	Type	Description
`members`	string	One or more named components, or the wildcard character (`*`) to retrieve all custom metadata components of the type specified in the `<name>` element. To retrieve a standard object, specify it by name. For example `<members>Account</members>` will retrieve the standard Account object.
`name`	string	The type of metadata component to be retrieved. For example `<name>CustomObject</name>` will retrieve one or more custom objects as specified in the `<members>` element.

`checkRetrieveStatus()`

Checks the status of declarative metadata call `retrieve()` and returns the zip file contents.

Syntax

```
RetrieveResult = metadatabinding.checkRetrieveStatus(ID id);
```

Usage

`checkRetrieveStatus` is part of the procedure for retrieving metadata components from an organization. It is used together with the `checkStatus` call which indicates when the asynchronous `retrieve` call has completed. Once `checkStatus` indicates that the call is completed, call `checkRetrieveStatus` to get the zip file contents:

1. Issue a `retrieve()` call to start the asynchronous retrieval. An AsyncResult object is returned. If the call is completed, the `done` field contains `true`. Most often, the call is not completed quickly enough to be noted in the result. If it is completed, note the value in the `id` field returned and skip the next step.
2. If the call is not complete, issue a `checkStatus()` call in a loop using the value in the `id` field of the AsyncResult object returned by the `retrieve()` call in the previous step. Check the AsyncResult object returned, until the `done` field contains `true`. The time taken to complete a `retrieve()` call depends on the size of the zip file being deployed, so a longer wait time between iterations should be used as the size of the zip file increases.
3. Issue a `checkRetrieveStatus()` call to obtain the results of the `retrieve()` call, using the `id` value returned in the first step.

Sample Code—Java

See the `retrieve()` sample code for sample usage of this call.

Arguments

Name	Type	Description
`id`	ID	ID obtained from a RetrieveResult object returned by a `retrieve()` call or a subsequent AsyncResult object returned by a `checkStatus()` call.

Response

RetrieveResult

Chapter 6

CRUD-Based Calls

Use the following CRUD-based calls to work with metadata components in a manner similar to the way synchronous API calls in the enterprise WSDL act upon objects.

- `create()`
- `update()`
- `delete()`

`create()`

Adds one or more new metadata components to your organization's data. This call can be used to create any of the objects that extend Metadata. For more details, see Metadata Types on page 71.

Syntax

```
AsyncResult[] = metadatabinding.create(Metadata[] metadata);
```

Usage

Use this call to add one or more metadata components to your organization's information.

Permissions

Your client application must be logged in with the "Modify All Data" permission.

Required Fields

Required fields are determined by the metadata components being created. For more information about specific component types, see Metadata Types on page 71.

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 `label` field is entered as `"MyObject "`, the value is stored in the database as `"MyObject"`.

Basic Steps for Creating Metadata Components

Use the following process to create metadata components:

1. Design an array and populate it with the components that you want to create.
2. Call `create()` with the component array passed in as an argument.
3. An AsyncResult object is returned for each component you tried to create. It is updated with status information as the operation moves from a queue to completed or error state. Call `checkStatus()` in a loop until the status values in AsyncResult indicate that all the create operations are completed. Start with a wait time of one second between iterations of `checkStatus()` calls, and double the wait time each time you make a subsequent call.

Sample Code—Java

See Step 4: Walk Through the Sample Code for sample Java code using the `create()` call.

Arguments

Name	Type	Description
`metadata`	Metadata[]	Array of one or more metadata components. Limit: 10. You must submit arrays of only one type of component. For example, you could submit an array of 10 custom objects or 10 profiles, but not a mix of both types.

Response

AsyncResult[]

delete()

Deletes one or more components from your organization's data. This call can be used to delete any of the objects that extend Metadata. For more details, see Metadata Types on page 71.

Syntax

```
AsyncResult[] = metadatabinding.delete(Metadata[] metadata);
```

Usage

Use this call to delete one or more components from your organization's data.

Permissions

Your client application must be logged in with the "Modify All Data" permission.

Rules and Guidelines

When deleting components, consider the following rules and guidelines:

- Your client application must be logged in with sufficient access rights to delete individual components within the specified component. For more information, see "Factors that Affect Data Access" in the *Web Services API Developer's Guide*.

- In addition, you might also need permission to access this component's parent component.
- To ensure referential integrity, this call supports cascading deletions. If you delete a parent component, you delete its children automatically, as long as each child component can be deleted.
- Unlike some standard objects, all metadata components can be deleted.

Basic Steps for Deleting Metadata Components

Use the following process to delete metadata components:

1. Determine the `fullName` of each component that you want to delete. See Metadata for more details on the `fullName` field. You must delete only components of the same type in a single call.
2. Invoke this call, passing in the array of metadata components with `fullName` specified.
3. An AsyncResult object is returned for each component you tried to delete. It is updated with status information as the operation moves from a queue to completed or error state. Call `checkStatus()` in a loop until the status values in AsyncResult indicate that all the delete operations are completed. Start with a wait time of one second between iterations of `checkStatus()` calls, and double the wait time each time you make a subsequent call.

Sample Code—Java

```
public void deleteSample()
    throws Exception
{
    CustomObject co = new CustomObject();
    String name = "MyCustomObject";
    co.setFullName(name + "__c");
    AsyncResult[] ars = metadatabinding.delete(new Metadata[] { co });
    AsyncResult asyncResult = ars[0];
    // set initial wait time to one second in milliseconds
    long waitTimeMilliSecs = 1000;
    while (!asyncResult.isDone()) {
        Thread.sleep(waitTimeMilliSecs);
        // double the wait time for the next iteration
        waitTimeMilliSecs *= 2;
        asyncResult = metadatabinding.checkStatus(
                new String[] {asyncResult.getId()})[0];
        System.out.println("Status is: " + asyncResult.getState());
    }

    if (asyncResult.getState() != AsyncRequestState.Completed) {
        throw new Exception(asyncResult.getStatusCode() + " msg: " +
                asyncResult.getMessage());
    }
}
```

Arguments

Name	Type	Description
`metadata`	Metadata[]	Array of one or more metadata components. You only need to set the `fullName` field in the Metadata object. Limit: 10. You must submit arrays of only one type of component. For example, you could submit an array of 10 custom objects or 10 profiles, but not a mix of both types.

Response

AsyncResult[]

update()

Updates one or more components in your organization's data. This call can be used to update any of the objects that extend Metadata. For more details, see Metadata Types on page 71.

Syntax

```
AsyncResult[] = metadatabinding.update(UpdateMetadata[] metadata);
```

Usage

Use this call to update one or more components. This call is analogous to the ALTER TABLE statement in SQL.

Permissions

Your client application must be logged in with the "Modify All Data" permission.

Updateable Objects

Unlike standard objects, all metadata components can be updated.

Required Fields

You must supply values for all the required fields in the component.

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 `label` field is entered as "   MyObject ", the value is stored in the database as "MyObject".

Basic Steps for Updating Metadata Components

Use this process to update metadata components:

1. Invoke this call, passing in an array of metadata components that represent the components you wish to update.
2. An AsyncResult object is returned for each component or field you tried to update. It is updated with status information as the operation moves from a queue to completed or error state. Use `checkStatus()` to check on the status values in AsyncResult.
3. An AsyncResult object is returned for each component you tried to update. It is updated with status information as the operation moves from a queue to completed or error state. In a loop, call `checkStatus()` until the status values in AsyncResult indicate that all the update operations are completed. Start with a wait time of one second between iterations of `checkStatus()` calls, and double the wait time each time you make a subsequent call.

Sample Code—Java

```
public void updateCustomObjectSample()
    throws Exception
{
    CustomObject co = new CustomObject();
    String name = "MyCustomObject";
    co.setFullName(name + "__c");
    co.setDeploymentStatus(DeploymentStatus.Deployed);
    co.setDescription("Created by the Metadata API");
    co.setEnableActivities(true);
    co.setLabel(name + " Object");
    co.setPluralLabel(co.getLabel() + "s");
    co.setSharingModel(SharingModel.ReadWrite);

    CustomField nf = new CustomField();
    nf.setType(FieldType.Text);
    nf.setLabel(co.getFullName() + " Name");

    co.setNameField(nf);

    UpdateMetadata updateMetadata = new UpdateMetadata();
    updateMetadata.setMetadata(co);
    updateMetadata.setCurrentName("TheCurrentName");   // co.getFullName();

    AsyncResult[] ars =
        metadatabinding.update(new UpdateMetadata[] { updateMetadata});
    AsyncResult asyncResult = ars[0];
    // set initial wait time to one second in milliseconds
    long waitTimeMilliSecs = 1000;
    while (!asyncResult.isDone()) {
        Thread.sleep(waitTimeMilliSecs);
        // double the wait time for the next iteration
        waitTimeMilliSecs *= 2;
        asyncResult = metadatabinding.checkStatus(
                new String[] {asyncResult.getId()})[0];
        System.out.println("Status is: " + asyncResult.getState());
    }

    if (asyncResult.getState() != AsyncRequestState.Completed) {
        throw new Exception(asyncResult.getStatusCode() + " msg: " +
                asyncResult.getMessage());
    }
}
```

Arguments

Name	Type	Description
`metadata`	UpdateMetadata[]	Array of one or more UpdateMetadata data structures that represent the components you wish to update. Limit: 10. You must submit arrays of only one type of component. For example, you could submit an array of 10 custom objects or 10 profiles, but not a mix of both types.

UpdateMetadata

One or more UpdateMetadata objects are defined in the `metadata` argument. This object can be used to update any of the objects that extend Metadata. For more details, see Metadata Types on page 71. Each UpdateMetadata object has the following fields:

Field	Field Type	Description
`currentName`	string	The API name of the component or field before the update. For example, if you wanted to update a CustomObject named Foo, the value of this field would be `Foo__c`. This value is supplied because this call may change the name, and the value here provides mapping.
`metadata`	Metadata	Full specification of the component or field you wish to update.

Response

AsyncResult[]

Chapter 7

Utility Calls

Use the following utility calls to gather information that is useful for working with the file-based or CRUD-based calls.

- `checkStatus()`
- `describeMetadata()`
- `listMetadata()`

`checkStatus()`

Checks the status of asynchronous metadata calls `create()`, `update()`, or `delete()`, or the declarative metadata calls `deploy()` or `retrieve()`.

Syntax

```
AsyncResult[] = metadatabinding.checkStatus(ID[] ids);
```

Usage

Use this call to check whether or not an asynchronous metadata call or declarative metadata call has completed.

Sample Code—Java

See Step 4: Walk Through the Sample Code for sample Java code using this call.

Arguments

Name	Type	Description
`ids`	ID[]	Array of one or more IDs. Each ID is returned in an AsyncResult and corresponds to a component being created, updated, deleted, deployed, or retrieved.

Response

AsyncResult[]

describeMetadata()

This call retrieves the metadata which describes your organization. This information includes Apex classes and triggers, custom objects, custom fields on standard objects, tab sets that define an app, and many other components.

Syntax

```
DescribeMetadataResult[] = metadatabinding.describeMetadata(double apiVersion);
```

Arguments

Name	Type	Description
`apiVersion`	double	The API version for which you want metadata; for example, 20.0.

Permissions

Your client application must be logged in with the “Modify All Data” permission.

Sample Code—Java

```
 private void describeMetadata()
    throws Exception
{
    try {
        double apiVersion = 20.0;
        // Assuming that the SOAP binding has already been established.
        DescribeMetadataResult res = metadatabinding.describeMetadata(apiVersion);
        StringBuffer sb = new StringBuffer();
        if (res != null && res.getMetadataObjects().length > 0) {
            for (DescribeMetadataObject obj : res.getMetadataObjects()) {
                sb.append("***************************************************\n");
                sb.append("XMLName: " + obj.getXmlName() + "\n");
                sb.append("DirName: " + obj.getDirectoryName() + "\n");
                sb.append("Suffix: " + obj.getSuffix() + "\n");
                sb.append("***************************************************\n");
            }
        } else {
            sb.append("Failed to obtain metadata types.");
        }
        System.out.println(sb.toString());
    }
    catch (Exception ex) {
    System.out.println("\nFailed to describe metadata, error message was: \n"
     + ex.getMessage());
    throw ex;
    }
}
```

Response

DescribeMetadataResult

listMetadata()

This call retrieves property information about metadata components in your organization. Data is returned for the components that match the criteria specified in the `queries` parameter. The `queries` array can contain up to three ListMetadataQuery queries for each call. This call supports every metadata type: both top-level, such as CustomObject and ApexClass, and child types, such as CustomField and RecordType.

Syntax

```
 FileProperties[] = metadatabinding.listMetadata(ListMetadataQuery[] queries, double
asOfVersion);
```

Usage

This call is useful when you want to identify individual components in `package.xml` for a `retrieve()` call or if you want a high-level view of particular metadata types in your organization. For example, you could use this call to return a list of names of all the CustomObject or Layout components in your organization, and use this information to make a subsequent `retrieve()` call to return a subset of these components. For more information about working with `package.xml`, see Deploying and Retrieving Metadata on page 25.

Note: This is a synchronous call so the results are returned in one call. This differs from asynchronous calls, such as `retrieve()`, where at least one subsequent call is needed to get the results.

Permissions

Your client application must be logged in with the “Modify All Data” permission.

Sample Code—Java

The sample code below lists information about your custom objects. The code assumes that the SOAP binding has already been established.

```
private void listMetadata()
    throws Exception
{
    try {
        String component = "CustomObject";
        String optionalFolder = null;
        ListMetadataQuery query = new ListMetadataQuery(optionalFolder, component);
        double asOfVersion = 20.0;

        // Assuming that the SOAP binding has already been established.
        FileProperties[] lmr = metadatabinding.listMetadata(
            new ListMetadataQuery[] {query}, asOfVersion);
        if (lmr != null) {
            for (FileProperties n : lmr) {
                System.out.println("Component fullName: " + n.getFullName());
                System.out.println("Component type: " + n.getType());
            }
        }
    }
    catch (Exception ex) {
        System.out.println("\nFailed to list metadata, error message was: \n"
             + ex.getMessage());
        throw ex;
    }
}
```

Arguments

Name	Type	Description
`queries`	ListMetadataQuery[]	A list of objects that specify which components you are interested in.
`asOfVersion`	double	The API version for the metadata listing request. If you don't specify a value in this field, it defaults to the API version specified when you logged in. This field allows you to override the default and set another API version so that, for example, you could list the metadata for a metadata type that was added in a later version than the API version specified when you logged in. This field is available in API version 18.0 and later.

Response

FileProperties

ListMetadataQuery

The ListMetadataQuery parameter specified in a `listMetadata()` call consists of the following properties:

Name	Type	Description
`folder`	string	The folder associated with the component. This field is required for components that use folders, such as Dashboard, Document, EmailTemplate, or Report.
`type`	string	Required. The metadata type, such as `CustomObject`, `CustomField`, or `ApexClass`.

Chapter 8

Result Objects

Use the following objects to get the results of your file-based or CRUD-based calls.

- AsyncResult
- DeployResult
- DescribeMetadataResult
- RetrieveResult

AsyncResult

Poll the values in this object to determine when an asynchronous metadata call has completed, and whether it was successful or not. The asynchronous metadata calls `create()`, `update()`, and `delete()` return an array of AsyncResult objects. Each element in the array corresponds to an element in the array of metadata components passed in the call.

Use the `checkStatus()` call against each object to discover when the call is completed for that object. Salesforce.com updates each AsyncResult object as the call completes, or when any errors occur.

The `deploy()` and `retrieve()` calls use AsyncResult similarly, though you must subsequently use `checkDeployStatus()` or `checkRetrieveStatus()` respectively to get more status information for the deployment or retrieval.

Each AsyncResult object has the following properties:

Name	Type	Description
`checkOnly`	boolean	Indicates whether this deployment is being used to check the validity of the deployed files without making any changes in the organization (`true`) or not (`false`). A check-only deployment does not deploy any components or change the organization in any way. This field is available in API version 16.0 and later and is only relevant for the `deploy()` call.
`done`	boolean	Required. Indicates whether the call has completed (`true`) or not (`false`).
`id`	ID	Required. ID of the component being created, updated, deleted, deployed, or retrieved.
`message`	string	Message corresponding to the `statusCode` field returned, if any.
`numberComponentErrors`	int	The number of components that generated errors during this deployment. This field is available in API version 16.0 and later and is only relevant for the `deploy()` call.
`numberComponentsDeployed`	int	The number of components that have been deployed so far for this deployment. This field in conjunction with the

Name	Type	Description
		`numberComponentsTotal` field gives you an indication of the progress of the deployment. This field is available in API version 16.0 and later and is only relevant for the `deploy()` call.
`numberComponentsTotal`	int	The total number of components in the deployment. This field in conjunction with the `numberComponentsDeployed` field gives you an indication of the progress of the deployment. This field is available in API version 16.0 and later and is only relevant for the `deploy()` call.
`numberTestErrors`	int	The number of Apex tests that have generated errors during this deployment. This field is available in API version 16.0 and later and is only relevant for the `deploy()` call.
`numberTestsCompleted`	int	The number of Apex tests that have completed so far for this deployment. This field in conjunction with the `numberTestsTotal` field gives you an indication of the progress of tests for the deployment. This field is available in API version 16.0 and later and is only relevant for the `deploy()` call.
`numberTestsTotal`	int	The total number of Apex tests in the deployment. This field in conjunction with the `numberTestsCompleted` field gives you an indication of the progress of tests for the deployment. The value in this field is not accurate until the deployment has started running tests for the components being deployed. This field is available in API version 16.0 and later and is only relevant for the `deploy()` call.
`secondsToWait`	int	This field is no longer supported for API version 13.0 and later and is only provided for backward compatibility. The field was removed in API version 17.0. Indicates the number of seconds before the call is likely to complete. This is an estimate only. A reasonable approach is to wait one second before calling `checkStatus()` to see if the operation is complete. Double your wait time for each successive iteration of `checkStatus()` calls until the operation is complete.
`state`	AsyncRequestState (enumeration of type string)	Required. The `AsyncRequestState` object has one of four possible values: • `Queued`: This call has not started. It is waiting in a queue. • `InProgress`: This call has started, but has not completed yet. • `Completed`: This call has completed. • `Error`: An error occurred, see the `statusCode` for more information.
`stateDetail`	string	Indicates which component is currently being deployed or which Apex test class is running. This field is available in API version 16.0 and later and is only relevant for the `deploy()` call.
`stateDetailLastModifiedDate`	dateTime	The data and time when the `stateDetail` field was last modified. This field is available in API version 16.0 and later and is only relevant for the `deploy()` call.

Name	Type	Description
`statusCode`	StatusCode (enumeration of type string)	If an error occurred during the `create()`, `update()`, or `delete()` call, a status code is returned, and the message corresponding to the status code is returned in the `message` field. For a description of each StatusCode value, see "StatusCode" in the *Web Services API Developer's Guide*.

DeployResult

The asynchronous metadata call `checkDeployStatus()` returns a DeployResult object, which contains information about the success or failure of the associated `deploy()` call:

Name	Type	Description
`id`	ID	ID of the component being deployed.
`messages`	DeployMessage[]	Contains information about the success or failure of a `deploy()` call.
`retrieveResult`	RetrieveResult	If the `performRetrieve` parameter was specified for the `deploy()`, a `retrieve()` is performed immediately after the `deploy()` is completed. This field contains the results of that retrieval.
`runTestResult`	RunTestsResult	If the `runAllTests` or `runTests` parameters are set to run tests, this field contains the results of those tests.
`success`	boolean	Indicates whether the deployment was successful (`true`) or not (`false`).

Usage

Contains information about the success or failure of a `deploy()` call.

DeployMessage

Each DeployResult object contains one or more DeployMessage objects. Each DeployMessage object contains information about the deployment success or failure of a component in the deployment `.zip` file:

Name	Type	Description
`changed`	boolean	If `true`, the component was changed as a result of this deployment. If `false`, the deployed component was the same as the corresponding component already in the organization.
`columnNumber`	int	Each component is represented by a text file. If an error occurred during deployment, this field represents the column of the text file where the error occurred.
`created`	boolean	If `true`, the component was created as a result of this deployment. If `false`, the component was either deleted or modified as a result of the deployment.
`deleted`	boolean	If `true`, the component was deleted as a result of this deployment. If `false`, the component was either new or modified as result of the deployment.

Name	Type	Description
fileName	string	The name of the file in the .zip file used to deploy this component.
fullName	string	Required. The full name of the component. Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See create() to see an example of this field specified for a call.
id	ID	ID of the component being deployed.
lineNumber	int	Each component is represented by a text file. If an error occurred during deployment, this field represents the line number of the text file where the error occurred.
problem	string	If an error or warning occurred, this field contains a description of the problem that caused the compile to fail.
problemType	DeployProblemType (enumeration of type string)	Indicates the problem type. The problem details are tracked in the problem field. The valid values are: • Warning • Error This field is available in API version 18.0 and later. Prior to version 18.0, there was no distinction between warnings and errors. All problems were treated as errors and prevented a successful deployment.
success	boolean	Indicates whether the component was successfully deployed (true) or not (false).

RunTestsResult

The call returns information about whether or not the compilation of the specified Apex was successful and if the unit tests completed successfully.

A RunTestsResult object has the following properties:

Name	Type	Description
codeCoverage	CodeCoverageResult[]	An array of one or more CodeCoverageResult objects that contains the details of the code coverage for the specified unit tests.
codeCoverageWarnings	CodeCoverageWarning[]	An array of one or more code coverage warnings for the test run. The results include both the total number of lines that could have been executed, as well as the number, line, and column positions of code that was not executed.
failures	RunTestFailure[]	An array of one or more RunTestFailure objects that contain information about the unit test failures, if there are any.
numFailures	int	The number of failures for the unit tests.
numTestsRun	int	The number of unit tests that were run.

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

CodeCoverageResult

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

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 covered.
namespace	string	The namespace that contained the unit tests, if one is specified.
numLocations	int	The number of locations covered.
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.
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 CodeLocation. The ID is unique within an organization.
message	string	The message of the warning generated.
name	string	The namespace that contained the unit tests, if one is specified.
namespace	string	The namespace that contained the unit tests, if one is specified.

RunTestFailure

The RunTestsResult object returns information about failures during the unit test run.

This object has the following properties:

Name	Type	Description
id	ID	The ID of the class which generated failures.
message	string	The failure message.
methodName	string	The name of the method that failed.
name	string	The name of the class that failed.
namespace	string	The namespace that contained the class, if one was specified.
stackTrace	string	The stack trace for the failure.
time	double	The time spent running tests for this failed operation. This can be helpful for performance monitoring.
type	string	Do not use. In early, unsupported releases, used to specify class or package.

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	Type	Description
`name`	string	The name of the class that succeeded.
`namespace`	string	The namespace that contained the unit tests, if one is specified.
`time`	double	The time spent running tests for this operation. This can be helpful for performance monitoring.

CodeLocation

The RunTestsResult object contains this object in a number of fields.

This object has the following properties:

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

DescribeMetadataResult

The call `describeMetadata()` returns information about the organization that is useful for developers working with declarative metadata.

Each DescribeMetadataResult object has the following properties:

Name	Type	Description
`metadataObjects`	DescribeMetadataObject[]	One or more metadata components and their attributes.
organizationNamespace	string	The namespace of the organization. Specify only for Developer Edition organizations that can contain a managed package. The managed package has a namespace specified when it is created.
`partialSaveAllowed`	boolean	Indicates whether `rollbackOnError` is allowed (`true`) or not (`false`). This value is always : • `false` in production organizations. • the opposite of `testRequired`.

Name	Type	Description
`testRequired`	boolean	Indicates whether tests are required (`true`) or not (`false`). This value is always the opposite of `partialSaveAllowed`.

DescribeMetadataObject

This object is returned as part of the DescribeMetadataResult. Each DescribeMetadataObject has the following properties:

Name	Type	Description
`childXmlNames`	string[]	List of child sub-components for this component.
`directoryName`	string	The name of the directory in the `.zip` file that contains this component.
`inFolder`	boolean	Indicates whether the component is in a folder (`true`) or not (`false`). For example, documents, email templates and reports are stored in folders.
`metaFile`	boolean	Indicates whether the component requires an accompanying metadata file. For example, documents, classes, and s-controls are components that require an additional metadata file.
`suffix`	string	The file suffix for this component.
`xmlName`	string	The name of the root element in the metadata file for this component. This name also appears in the `Packages > types > name` field in the manifest file `package.xml`.

RetrieveResult

The metadata call `retrieve()` returns a RetrieveResult object, which contains information about the success or failure of the associated `retrieve()` call.

Each RetrieveResult object has the following fields:

Name	Type	Description
fileProperties	FileProperties[]	Contains information about the properties of each component in the `.zip` file, and the manifest file `package.xml`. One object per component is returned.
id	ID	ID of the component being retrieved.
messages	RetrieveMessage[]	Contains information about the success or failure of the `retrieve()` call.
zipFile	base64Binary	The zip file returned by the retrieve request. Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion is usually handled for you by a SOAP client.

FileProperties

This component contains information about the properties of each component in the `.zip` file, and the manifest file `package.xml`. One object per component is returned. Note that this component does not contain information about any associated metadata files in the `.zip` file, only the component files and manifest file. FileProperties contains the following properties:

Name	Type	Description
`createdById`	string	Required. ID of the user who created the file.
`createdByName`	string	Required. Name of the user who created the file.
`createdDate`	dateTime	Required. Date and time when the file was created.
`fileName`	string	Required. Name of the file.
`fullName`	string	Required. The file developer name used as a unique identifier for API access. The value is based on the `fileName` but the characters allowed are more restrictive. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
`id`	string	Required. ID of the file.
`lastModifiedById`	string	Required. ID of the user who last modified the file.
`lastModifiedByName`	string	Required. Name of the user who last modified the file.
`lastModifiedDate`	dateTime	Required. Date and time that the file was last modified.
`manageableState`	ManageableState, (enumeration of type string)	Indicates the manageable state of the specified component if it is contained in a package: • `beta` • `deleted` • `deprecated` • `installed` • `released` • `unmanaged` For more information about states of manageability for components in Force.com AppExchange packages, see "Planning the Release of Managed Packages" in the Salesforce.com online help.
`namespacePrefix`	string	If any, the namespace prefix of the component.
`type`	string	Required. The metadata type, such as `CustomObject`, `CustomField`, or `ApexClass`.

RetrieveMessage

RetrieveResult returns this object, which contains information about the success or failure of the `retrieve()` call. One object per problem is returned:

Name	Type	Description
`fileName`	string	The name of the file in the retrieved `.zip` file where a problem occurred.
`problem`	string	A description of the problem that occurred.

Chapter 9

Metadata Types

Metadata components are not based on sObjects, like objects in the API. Instead, they are based on metadata types, such as ApexClass and CustomObject, which extend Metadata. A component is an instance of a metadata type. For example, `CustomObject` is a metadata type for custom objects, and the `MyCustomObject__c` component is an instance of a custom object.

A metadata type can be identified in the metadata WSDL as any complexType that extends the Metadata complexType. A complexType that is a metadata type includes the following element in its WSDL definition:

```
<xsd:extension base="tns:Metadata">
```

CustomObject and BusinessProcess extend Metadata so they are metadata types; ActionOverride doesn't extend Metadata so it's not a metadata type.

You can individually deploy or retrieve a component for a metadata type. For example, you can retrieve an individual BusinessProcess component, but you can't retrieve an individual ActionOverride component. You can only retrieve an ActionOverride component by retrieving its encompassing CustomObject component.

Metadata components can be manipulated by asynchronous Metadata API calls or declarative (or file-based) Metadata API calls.

Most of the components can be accessed using Force.com IDE. Exceptions are noted in the description of the object.

Field Data Types

Each component field has a specific field type. These field types can correspond to other components defined in the WSDL, or primitive data types, like `string`, that are commonly used in strongly typed programming languages.

These field data types are used in the SOAP messages that are exchanged between your client application and the API. When writing your client application, follow the data typing rules defined for your programming language and development environment. Your development tool handles the mapping of typed data in your programming language with these SOAP data types.

For more information about primitive data types, see the *Web Services API Developer's Guide*.

Enumeration Fields

Some component fields have a data type that is an enumeration. An enumeration is the API equivalent of a picklist. The valid values of the field are restricted to a strict set of possible values, all having the same data type. These values are listed in the field description column for each enumeration field. See sortBy for an example of an enumeration field of type string. The XML below shows a sample definition of an enumeration of type string in the WSDL.

```
<xsd:simpleType name="DashboardComponentFilter">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="RowLabelAscending"/>
        <xsd:enumeration value="RowLabelDescending"/>
        <xsd:enumeration value="RowValueAscending"/>
        <xsd:enumeration value="RowValueDescending"/>
    </xsd:restriction>
</xsd:simpleType>
```

Supported Calls

All of the metadata types are supported by the main calls, unless it is stated otherwise in the individual component sections. The main Metadata API calls are `create()`, `delete()`, `update()`, `deploy()`, `retrieve()`, `listMetadata()`, and `describeMetadata()`. All other calls, such as `checkStatus()`, are considered utility calls as they are used in conjunction with one of the main calls.

AnalyticSnapshot

An analytic snapshot lets you report on historical data. Authorized users can save tabular or summary report results to fields on a custom object, then map those fields to corresponding fields on a target object. They can then schedule when to run the report to load the custom object's fields with the report's data. Analytic snapshots let you to work with report data similarly to how you work with other records in Salesforce.com.

Declarative Metadata File Suffix and Directory Location

Force.com AnalyticSnapshot components are stored in the `analyticSnapshots` directory of the corresponding package directory. The file name matches the unique name of the analytic snapshot, and the extension is `.analyticsnapshot`.

Version

Force.com AnalyticSnapshot components are available in API version 16.0 and later.

Fields

Field	Field Type	Description
`description`	string	A description of the analytic snapshot.
`fullName`	string	The analytic snapshot name used for API access. The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. This field is inherited from the Metadata component.
`groupColumn`	string	A column that specifies which level to extract data from the source report. It is only applicable for summary reports.

Field	Field Type	Description
`mappings`	AnalyticSnapshotMapping[]	A list of analytic snapshot mappings. For valid values, see AnalyticSnapshotMapping.
`name`	string	Required. The display name of the analytic snapshot.
`runningUser`	string	The username of the user whose role and *sharing* settings are used to run the analytic snapshot.
`sourceReport`	string	Required. The report where data will be extracted from.
`targetObject`	string	Required. The custom object where data will be inserted into.

AnalyticSnapshotMapping

AnalyticSnapshotMapping defines the mapping for the analytic snapshot. Valid values are:

Field	Field Type	Description
`aggregateType`	ReportSummaryType[] (enumeration of type string)	List that defines if and how each report field is summarized. For valid values, see ReportSummaryType.
`sourceField`	string	The sourceField can be one of the following: • The field on the sourceReport that you want to map to the targetField in the targetObject • A summary of a filed on the sourceReport (for Summary reports only) • A field on the analytic snapshot, such as JobName, RunningUser, or ExecutionTime (set through the user interface) **Note:** The sourceField must correspond to the sourceType you specify.
`sourceType`	ReportJobSourceTypes[] (enumeration of type string)	List that defines the report format for the analytic snapshot. For valid values, see ReportJobSourceTypes.
`targetField`	string	A field on the targetObject into which this particular sourceField will be inserted.

ReportJobSourceTypes

An enumeration of type string that defines the report format for the analytic snapshot. Valid values are:

Enumeration Value	Description
`snapshot`	Use this option if the sourceField contains snapshot-specific information such as JobName, RunningUser, or ExecutionTime.
`summary`	Use this option if referencing a summary (Sum, Average, Minimum, Maximum) of a field from the sourceReport.
`tabular`	Use this option if referencing an available column from the sourceReport.

Declarative Metadata Sample Definition

A sample XML definition of an analytic snapshot is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<AnalyticSnapshot xmlns="http://soap.sforce.com/2006/04/metadata">
    <description>my description</description>
    <groupColumn>INDUSTRY</groupColumn>
    <mappings>
        <aggregateType>Average</aggregateType>
        <sourceField>SALES</sourceField>
        <sourceType>summary</sourceType>
        <targetField> myObject __c.Name</targetField>
    </mappings>
    <mappings>
        <sourceField>ExecutionTime</sourceField>
        <sourceType>snapshot</sourceType>
        <targetField> myObject __c.field3__c</targetField>
    </mappings>
    <mappings>
        <sourceField>INDUSTRY</sourceField>
        <sourceType>tabular</sourceType>
        <targetField>testObject__c.Name</targetField>
    </mappings>
    <name>my snapshot</name >
    <runningUser>user@salesforce.com</runningUser>
    <sourceReport>myFolder/mytSummaryReport</sourceReport>
    <targetObject>myObject__c</targetObject>
</AnalyticSnapshot>
```

See Also:

Report

ArticleType

Represents the metadata associated with an article type. All articles in Salesforce Knowledge are assigned to an *article type* depending on their content. For example, a simple FAQ article type may have two custom fields, `Question` and `Answer`, where article managers enter data when creating or updating FAQ articles. A more complex article type may require dozens of fields organized into several sections. Using layouts and templates, administrators can structure the article type in the most effective way for its particular content. See “Managing Article Types” in the Salesforce.com online help and “Articles” in the Web Services API Developer's Guide.

Declarative Metadata File Suffix and Directory Location

An ArticleType is defined as a custom object and is stored in the `objects` folder. ArticleTypes have a suffix `__kav` (instead of `__c` for custom objects). ArticleType field names have a suffix of `__c` like other custom objects, and must be dot-qualified with the name of the article type to which they belong. This is shown in the following sample `package.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <fullName>articlefilemetadata</fullName>
    <apiAccessLevel>Unrestricted</apiAccessLevel>

    <types>
        <members>newarticle__kav.description__c</members>
        <name>CustomField</name>
    </types>
    <types>
        <members>newarticle__kav</members>
        <name>CustomObject</name>
    </types>

</Package>
```

Version

ArticleTypes are available in API version 19.0 and later.

Fields

Field Name	Field Type	Description
`articleTypeChannelDisplay`	articleTypeChannelDisplay	Represents the article-type templates used to display an article in the various channels. See "Assigning Article-Type Templates" in the Salesforce.com online help.
`deploymentStatus`	DeploymentStatus (enumeration of type string)	A string which represents the deployment status of a custom object or field. Valid values are: • `InDevelopment` • `Deployed`
`description`	string	A description of the article type. Maximum of 1000 characters.
`fields`	CustomField[]	Represents one or more fields in the article type.
`gender`	Gender	Gender of the name to support translation for languages that indicate gender in nouns. Valid values are: • `Neuter` • `Masculine` • `Feminine`
`label`	string	Label that represents the object throughout the Salesforce.com user interface.
`pluralLabel`	string	Plural version of the `label` value.
`startsWith`	StartsWith (enumeration of type string)	Indicates whether the name starts with a vowel, consonant, or is a special character. This is used for languages where words need different treatment depending on the first character. Valid values are listed in StartsWith.

articleTypeChannelDisplay

Determines the article-type templates that are used to display an article in its channels. Unless otherwise noted, all fields are createable, filterable, and nillable.

Field Name	Field Type	Description
`articleTypeTemplates`	articleTypeTemplates	Indicates which article-type template applies in the specified channel.

articleTypeTemplates

Sets the article-type template for a specific channel. If not specified, the default article-type template applies.

Field Name	Field Type	Description
`channel`	string	Specifies the channel where the article-type template applies: • `AllChannels`: all the available channels. • `App`: the Articles tab in Salesforce Knowledge. • `Pkb`: the public knowledge base. • `Csp`: the Customer Portal. • `Prm`: the partner portal. For more information about channels, see "Useful Salesforce Knowledge Terminology" in the Salesforce.com online help.
`page`	string	Represents the name of the custom Visualforce page used as a custom article-type template. Use this field when you select `Page` in the template field.
`template`	string	Indicates the article-type template used for the specified channel: • `Page`: custom Visualforce page. When specifying this value, you must also set the `page` field with the Visualforce page name. • `Tab`: display the sections you defined in the layout as tabs. • `Toc`: display the sections you defined in the layout as table of content.

Declarative Metadata Sample Definitions

A sample article type definition follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <articleTypeChannelDisplay>
        <articleTypeTemplates>
            <channel>App</channel>
            <template>Tab</template>
        </articleTypeTemplates>
        <articleTypeTemplates>
            <channel>Prm</channel>
            <template>Tab</template>
        </articleTypeTemplates>
        <articleTypeTemplates>
            <channel>Csp</channel>
            <template>Tab</template>
        </articleTypeTemplates>
        <articleTypeTemplates>
            <channel>Pkb</channel>
            <template>Toc</template>
        </articleTypeTemplates>
    </articleTypeChannelDisplay>
    <deploymentStatus>Deployed</deploymentStatus>
    <description>Article type with custom fields</description>
    <fields>
        <fullName>description__c</fullName>
        <label>Description</label>
        <length>48</length>
        <type>Text</type>
    </fields>
    <label>newarticle</label>
    <pluralLabel>newarticles</pluralLabel>
</CustomObject>
```

See Also:

ArticleType Layout
ArticleType CustomField

ArticleType Layout

Represents the metadata associated with an article type page layout. Article type layouts determine which fields users can view and edit when entering data for an article, they also determine which sections appear when users view articles. The format of the article, for example whether layout sections display as subtabs or as a single page with links, is defined by the article-type template. Each article type has only one layout, but you can choose a different template for each of the article type's four channels. For more information, see "Managing Article Types" in the Salesforce.com online help and "Articles" in the Web Services API Developer's Guide

File Suffix and Directory Location

ArticleType layouts are stored in the `layouts` directory of the corresponding package directory. The prefix must match with the article type API name. The extension is `.layout`.

Version

ArticleType layouts are available in API version 19.0 and later.

Fields

Field Name	Field Type	Description
`layoutSections`	LayoutSection[]	The main sections of the layout containing the article fields. The order here determines the layout order.

LayoutSection

LayoutSection represents a section of an ArticleType layout.

Field Name	Field Type	Description
`customLabel`	boolean	Indicates if this section's label is custom or standard (built-in). Custom labels can be any text, but must be translated. Standard labels have a predefined set of valid values, for example 'System Information', which are automatically translated.
`label`	string	The label; either standard or custom, based on the `customLabel` flag.
`layoutColumns`	LayoutColumn[]	The columns of the layout, depending on the style. Salesforce Knowledge only supports one column in article type layouts.
`style`	LayoutSectionStyle (enumeration of type string)	The style of the layout. Salesforce Knowledge only supports the value `OneColumn` which displays a one column page.

LayoutColumn

LayoutColumn represents the items in a column within a layout section.

Field Name	Field Type	Description
layoutItems	LayoutItem[]	The individual items within a column (ordered from top to bottom).

LayoutItem

LayoutItem represents the valid values that define a layout item.

Field Name	Field Type	Description
field	string	The field name reference, for example MyField__c.

Declarative Metadata Sample Definition

The following is the definition of an ArticleType page layout:

```
<?xml version="1.0" encoding="UTF-8"?>
<Layout xmlns="http://soap.sforce.com/2006/04/metadata">
    <layoutSections>
        <customLabel>true</customLabel>
        <label>Description</label>
        <layoutColumns>
            <layoutItems>
                <field>description__c</field>
            </layoutItems>
            <layoutItems>
                <field>dateTime__c</field>
            </layoutItems>
        </layoutColumns>
        <style>OneColumn</style>
    </layoutSections>
    <layoutSections>
        <label>Data Sheet</label>
        <layoutColumns>
            <layoutItems>
                <field>file__c</field>
            </layoutItems>
        </layoutColumns>
        <style>OneColumn</style>
    </layoutSections>
</Layout>
```

See Also:

ArticleType
ArticleType CustomField

ArticleType CustomField

Represents the metadata associated with an article type custom field. Use this metadata type to create, update, or delete article type custom field definitions. It extends the Metadata metadata type and inherits its fullName field.

You must specify the full name whenever you create or update a custom field. For example, a custom field on a custom object:

```
MyArticleType__kav.MyCustomField__c
```

Declarative Metadata File Suffix and Directory Location

Custom fields are defined as part of the article type. ArticleType field names have a suffix of `__c` like other custom objects, and must be dot-qualified with the name of the article type to which they belong. See ArticleType for more information.

Retrieving Custom Fields on Custom or Standard Objects

When you retrieve a custom or standard object, you return everything associated with the object. However, you can also retrieve only the custom fields for an object by explicitly naming the object and fields in `package.xml`. The following definition in `package.xml` will retrieve the files `objects/MyCustomObject__c.object`, `objects/Account.object__c.object` and `objects/MyArticleType__kav.object`, each containing one custom field definition.

```
<types>
  <members>MyCustomObject__c.MyCustomField__c</members>
  <members>Account.MyCustomAccountField__c</members>
  <members>MyArticleType__kav.MyOtherCustomField__c</members>
  <name>CustomField</name>
</types>
```

Version

ArticleTypes custom fields are available in API version 19.0 and later.

Fields for ArticleType

Unless otherwise noted, all fields are createable, filterable, and nillable.

Field Name	Field Type	Description
`description`	string	Description of the field.
`fullName`	string	Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call. This value cannot be `null`.
`inlineHelpText`	string	Represents the content of field-level help. For more information, see “Defining Field-Level Help” in the Salesforce.com online help.
`label`	string	Label for the field. You cannot update the label for standard fields in Article Type such as Title, UrlName, Summary, etc.
`length`	int	Length of the field.
`picklist`	Picklist	If specified, the field is a picklist, and this field enumerates the picklist values and labels.
`type`	FieldType	Required. Indicates the field type for the field. Valid values are: • `Date`

Field Name	Field Type	Description
		• `DateTime` • `Picklist` • `MultiselectPicklist` • `Text` • `TextArea` • `LongTextArea` • `File` • `Html`
`visibleLines`	int	Indicates the number of lines displayed for the field.

Declarative Metadata Sample Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
....
<fields>
        <fullName>Comments__c</fullName>
        <description>add your comments about this object here</description>
        <label>Comments</label>
        <length>32000</length>
        <type>LongTextArea</type>
        <visibleLines>30</visibleLines>
    </fields>
....
</CustomObject>
```

See Also:

ArticleType
ArticleType Layout

ApexClass

Represents an Apex class. An Apex class is 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. For more information, see the *Force.com Apex Code Developer's Guide*. This metadata type extends the MetadataWithContent component and shares its fields.

Supported Calls

`deploy()`, `retrieve()`, `describeMetadata()`, `listMetadata()`

Note: This metadata type is not supported by the `create()`, `delete()`, and `update()` calls.

Declarative Metadata File Suffix and Directory Location

The file suffix is `.cls` for the class file. The accompanying metadata file is named `*ClassName*-meta.xml`.

Apex classes are stored in the `classes` folder in the corresponding package directory.

Version

Apex classes are available in API version 10.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`apiVersion`	double	The API version for this class. Every class has an API version specified at creation.
`content`	base64	The Apex class definition. Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion is usually handled for you by a SOAP client. This field is inherited from the MetadataWithContent component.
`fullName`	string	The Apex class name. The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. This field is inherited from the Metadata component.
`packageVersions`	PackageVersion[]	The list of installed managed package versions that are referenced by this Apex class. For more information about managed packages, see the Force.com Quick Reference for Developing Packages. For more information about package versions, see "About Package Versions" in the Salesforce.com online help. This field is available in API version 16.0 and later.
`status`	ApexCodeUnitStatus (enumeration of type string)	The current status of the Apex class. The following string values are valid: • `Active` - The class is active. • `Deleted` - The class is marked for deletion. This is useful for managed packages, because it allows a class to be deleted when a managed package is updated. **Note:** ApexCodeUnitStatus includes an `Inactive` option, but it is only supported for ApexTrigger; it is not supported for ApexClass.

PackageVersion

PackageVersion identifies a version of a managed package. A package version is a number that identifies the set of components uploaded in a package. The version number has the format *`majorNumber.minorNumber.patchNumber`* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The *`patchNumber`* is generated and updated only for a patch release. If there is no *`patchNumber`*, it is assumed to be zero (0). Patch versions are currently available through a pilot program. For information on enabling patch versions for your organization, contact salesforce.com. It is available in API version 16.0 and later.

Field Name	Field Type	Description
`namespace`	string	Required. In a packaging context, a namespace prefix is a one to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other developers on AppExchange. Namespace prefixes are case-insensitive. For example, ABC and abc are not recognized as unique. Your namespace prefix must be globally unique across all Salesforce.com organizations. It keeps your managed package under your control exclusively. Salesforce.com automatically prepends your namespace prefix, followed by two underscores ("__"), to all unique component names in your Salesforce.com organization. A unique package component is one that requires a name that no other component has within Salesforce.com, such as custom objects, custom fields, custom links, s-controls, and validation rules. For more information about namespaces, see "Registering a Namespace Prefix" in the Salesforce.com online help.
`majorNumber`	int	Required. The major number of the package version. A package version number has a *`majorNumber.minorNumber`* format.
`minorNumber`	int	Required. The minor number of the package version. A package version number has a *`majorNumber.minorNumber`* format.

Declarative Metadata Sample Definition

The following sample creates the `MyhelloWorld.cls` class, and the corresponding `MyHelloWorld.cls-meta.xml` metadata file.

`MyHelloWorld.cls` file:

```
public class MyHelloWorld {
// This method updates the Hello field on a list
// of accounts.
public static void addHelloWorld(Account[] accs){
 for (Account a:accs){
  if (a.Hello__c != 'World')
  a.Hello__c = 'World';
  }
 }
}
```

`MyHelloWorld.cls-meta.xml:`

```
<?xml version="1.0" encoding="UTF-8"?>
<ApexClass xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>20.0</apiVersion>
</ApexClass>
```

See Also:

ApexTrigger

ApexComponent

Represents a Visualforce component. For more information, see “Visualforce Overview” in the Salesforce.com online help. This metadata type extends the MetadataWithContent component and shares its fields.

Declarative Metadata File Suffix and Directory Location

The file suffix is `.component` for the page file. The accompanying metadata file is named `ComponentName-meta.xml`.

Visualforce components are stored in the `components` folder in the corresponding package directory.

Version

Visualforce components are available in API version 12.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`apiVersion`	double	The API version for this Visualforce component. Every component has an API version specified at creation. This field is available in API version 16.0 and later.
`content`	base64Binary	The component content. Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion is usually handled for you by a SOAP client. This field is inherited from the MetadataWithContent component.
`description`	string	A description of what the component does.
`fullName`	string	The component developer name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`label`	string	Required. The label for this component.
`packageVersions`	PackageVersion[]	The list of installed managed package versions that are referenced by this Visualforce component. **Note:** Package components and custom component are distinct concepts. A package is comprised of many components, such as custom objects, Apex classes and triggers, and custom components. For more information about managed packages, see the Force.com Quick Reference for Developing Packages. For more information about package

Field Name	Field Type	Description
		versions, see “About Package Versions” in the Salesforce.com online help. This field is available in API version 16.0 and later.

See Also:

ApexPage

ApexPage

Represents a Visualforce page. For more information, see “Visualforce Overview” in the Salesforce.com online help. This metadata type extends the MetadataWithContent component and shares its fields.

Declarative Metadata File Suffix and Directory Location

The file suffix is `.page` for the page file. The accompanying metadata file is named *`PageName`*`-meta.xml`.

Visualforce pages are stored in the `pages` folder in the corresponding package directory.

Version

Visualforce pages are available in API version 11.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`apiVersion`	double	Required. The API version for this page. Every page has an API version specified at creation. This field is available in API version 15.0 and later. If you set this field to a number lower than 15.0, it will be changed to 15.0.
`content`	base64Binary	The page content. Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion is usually handled for you by a SOAP client. This field is inherited from the MetadataWithContent component.
`description`	string	A description of what the page does.
`fullName`	string	The page developer name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`label`	string	Required. The label for this page.

Field Name	Field Type	Description
`packageVersions`	PackageVersion[]	The list of installed managed package versions that are referenced by this Visualforce page. For more information about managed packages, see the Force.com Quick Reference for Developing Packages. For more information about package versions, see "About Package Versions" in the Salesforce.com online help. This field is available in API version 16.0 and later.

Declarative Metadata Sample Definition

The following sample creates the `MyPage.page` page, and the corresponding `MyPage.page-meta.xml` metadata file.

`SampleApexPage.page` file:

```
<apex:page>
<h1>Congratulations</h1>
This is your new Page.
</apex:page>
```

`SampleApexPage.page-meta.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<ApexPage xmlns="http://soap.sforce.com/2006/04/metadata">
    <description>This is a sample Visualforce page.</description>
    <label>SampleApexPage</label>
</ApexPage>
```

See Also:

ApexComponent

ApexTrigger

Represents an Apex trigger. A trigger is an Apex script that executes before or after specific data manipulation language (DML) events occur, such as before object records are inserted into the database, or after records have been deleted. For more information, see "Managing Apex Triggers " in the Salesforce.com online help. This metadata type extends the MetadataWithContent component and shares its fields.

Supported Calls

`deploy()`, `retrieve()`, `describeMetadata()`, `listMetadata()`

Note: This metadata type is not supported by the `create()`, `delete()`, and `update()` calls.

Declarative Metadata File Suffix and Directory Location

The file suffix is `.trigger` for the s-control file. The accompanying metadata file is named *`TriggerName`*`-meta.xml`.

Apex triggers are stored in the `triggers` folder in the corresponding package directory.

Version

Triggers are available in API version 10.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`apiVersion`	double	Required. The API version for this trigger. Every trigger has an API version specified at creation.
`content`	base64	The Apex trigger definition. This field is inherited from the MetadataWithContent component.
`fullName`	string	The Apex trigger name. The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. This field is inherited from the Metadata component.
`packageVersions`	PackageVersion[]	The list of installed managed package versions that are referenced by this Apex trigger. For more information about managed packages, see the Force.com Quick Reference for Developing Packages. For more information about package versions, see “About Package Versions” in the Salesforce.com online help. This field is available in API version 16.0 and later.
`status`	ApexCodeUnitStatus (enumeration of type string)	Required. The current status of the Apex trigger. The following string values are valid: • `Active` - The trigger is active. • `Inactive` - The trigger is inactive, but not deleted. • `Deleted` - The trigger is marked for deletion. This is useful for managed packages, because it allows a trigger to be deleted when a managed package is updated.

Declarative Metadata Sample Definition

The following sample creates the `MyhelloWorld.trigger` trigger, and the corresponding `MyHelloWorld.trigger-meta.xm` metadata file.

`MyHelloWorld.trigger` file:

```
Sample not yet available.
```

`MyHelloWorld.trigger-meta.xml:`

```
<?xml version="1.0" encoding="UTF-8"?>
<ApexTrigger xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>20.0</apiVersion>
</ApexTrigger>
```

See Also:

ApexClass

CustomApplication

An application is a list of tab references, with a description and a logo. CustomApplication represents a custom application. It extends the Metadata metadata type and inherits its `fullName` field.

File Suffix and Directory Location

File suffix: `.app`

Folder: `applications`

Note: Retrieving a component of this metadata type in a project makes the component appear in the Profile component as well.

Version

Custom applications are available in API version 10.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`defaultLandingPad`	string	The `fullName` of a standard tab or custom tab that opens when this application is selected.
`description`	string	The optional description text of the application.
`fullName`	string	The internal name of the application, based on the `label`, but with white spaces and special characters escaped out for validity. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`label`	string	Required. The name of the application.
`logo`	string	The optional reference to the image document for the application.
`tab`	string[]	The list of tabs included in this application. In API version 12.0, the `fullName` for built-in tabs like Home, Account, and Reports, is the name of the tab (Home, for example). In API version 13.0 and later, built-in tabs are prefixed with `std-` For example, to reference the Account tab you would use `std-Account`.

Declarative Metadata Sample Definition

The following is the definition of a custom application:

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomApplication xmlns="http://soap.sforce.com/2006/04/metadata">
    <description>App to manage Myriad Publishing</description>
    <logo>MyriadFolder/Myriad_Logo.jpg</logo>
    <tab>Campaign</tab>
    <tab>Lead</tab>
    <tab>Account</tab>
    <tab>Contact</tab>
    <tab>Myriad Publications</tab>
    <tab>Document</tab>
    <tab>report</tab>
</CustomApplication>
```

See Also:

CustomTab

CustomLabels

This metadata type allows you to create custom labels that can be localized for use in different languages, countries, and currencies. It extends the Metadata metadata type and inherits its `fullName` field. Custom labels are custom text values, up to 1,000 characters in length, that can be accessed from Apex classes or Visualforce pages. For more information, see “Custom Labels Overview” in the Salesforce.com online help.

Declarative Metadata File Suffix and Directory Location

Master custom label values are stored in the `CustomLabels.labels` file. Translations are stored in a file with a name format of `Translation-`*`localeCode`*`.translation`, where *`localeCode`* is the locale code of the translation language. The supported locale codes are listed in Language on page 205.

Custom label translations are stored in the `labels` folder in the corresponding package directory.

Version

CustomLabels components are available in API version 14.0 and later.

Fields

Field	Field Type	Description
`fullName`	string	Required. The name of the custom label bundle. Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`labels`	CustomLabel[]	A list of custom labels.

CustomLabel

This metadata type represents a custom label. It extends the Metadata metadata type and inherits its `fullName` field.

Field	Field Type	Description
`categories`	string	A comma-separated list of categories for the label. This field can be used in filter criteria when creating custom label list views. Maximum of 255 characters.

Field	Field Type	Description
`fullName`	string	Required. The name of the custom label. Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`language`	string	Required. The language of the translated custom label.
`protected`	boolean	Required. Indicates whether this component is protected (`true`) or not (`false`). Protected components cannot be linked to or referenced by components created in the installing organization.
`shortDescription`	string	Required. An easily recognizable term to identify this custom label. This description is used in merge fields.
`value`	string	Required. The translated custom label. Maximum of 1000 characters.

Declarative Metadata Sample Definition

A sample XML definition of a custom label component is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomLabels xmlns="http://soap.sforce.com/2006/04/metadata">
    <labels>
        <fullName>quoteManual</fullName>
        <label>This is a manual quote.</label>
        <language>en_US</language>
        <protected>false</protected>
        <shortDescription>Manual Quote</shortDescription>
    </labels>
    <labels>
        <fullName>quoteAuto</fullName>
        <label>This is an automatically generated quote.</label>
        <language>en_US</language>
        <protected>false</protected>
        <shortDescription>Automatic Quote</shortDescription>
    </labels>
</CustomLabels>
```

See Also:

Translations

CustomObject

Represents a custom object that stores data unique to your organization. It extends the Metadata metadata type and inherits its `fullName` field. You must specify all relevant fields when you create or update a custom object. You cannot update a single field on the object. For more information about custom objects, see “Custom Object Record Overview” in the Salesforce.com online help.

You can also use this metadata type to work with customizations of standard objects, such as accounts. For an example, see Standard Objects on page 29.

All metadata components have a `fullName` field, which must be fully specified for any custom object.

For example, the following are fully specified names:

```
Account
MyCustomObject__c
```

For sample Java code that creates a custom object, see Step 4: Walk Through the Sample Code on page 9 .

Declarative Metadata File Suffix and Directory Location

Custom object names are automatically appended with __c. The file suffix is `.object` for the custom object (or standard object) file.

Custom and standard objects are stored in the `objects` folder in the corresponding package directory.

Note: Retrieving a component of this metadata type in a project makes the component appear in the Profile component as well.

Version

Custom objects are available in API version 10.0 and later.

Fields

Unless otherwise noted, all fields are createable, filterable, and nillable.

Field Name	Field Type	Description
`actionOverrides`	ActionOverride[]	A list of action overrides on a standard or custom object. This field is available in API version 18.0 and later.
`businessProcesses`	BusinessProcess[]	A list of business processes associated with the object. This field is available in API version 17.0 and later.
`customHelp`	string	The s-control that contains the help content if this custom object has customized help content. This field is available in API version 14.0 and later.
`customHelpPage`	string	The Visualforce page that contains the help content if this custom object has customized help content. This field is available in API version 16.0 and later.
`customSettingsType`	CustomSettingsType (enumeration of type string)	When this field is present, this component is not a custom object, but a custom setting. This field returns the type of custom setting. The following string values are valid: • `List`—static data stored in cache and accessed as part of your application and available organization-wide. • `Hierarchy`—static data stored in cache and accessed as part of your application and available based on a hierarchy of user, profile or organization. This is the default value.

Field Name	Field Type	Description
		This field is available in API version 17.0 and later.
`customSettingsVisibility`	CustomSettingsVisibility (enumeration of type string)	When this field is present, this component is not a custom object, but a custom setting. This field returns the visibility of the custom setting. The following string values are valid: • `Public`—if the custom setting is packaged, it is accessible to all subscribing organizations. • `Protected`—if the custom setting is in a managed package, it is only accessible to the developer organization. Subscribing organizations cannot access it. This is the default value. This field is available in API version 17.0 and later.
`deploymentStatus`	DeploymentStatus (enumeration of type string)	Indicates the deployment status of the custom object.
`deprecated`	boolean	Reserved for future use.
`description`	string	A description of the object. Maximum of 1000 characters.
`enableActivities`	boolean	Indicates whether the custom object is enabled for activities (`true`) or not (`false`).
`enableDivisions`	boolean	Indicates whether the custom object is enabled for divisions (`true`) or not (`false`). For more information about the Division object, see the *Web Services API Developer's Guide*.
`enableEnhancedLookup`	boolean	Indicates whether the custom object is enabled for enhanced lookups (`true`) or not (`false`). Enhanced lookups provide an updated lookup dialog interface that gives users the ability to filter, sort, and page through search results as well as customize search result columns. For more information about enhanced lookups, see "Enabling Enhanced Lookups" in the Salesforce.com online help.
`enableFeeds`	boolean	Indicates whether the custom object is enabled for feed tracking (`true`) or not (`false`). For more information, see "Customizing Chatter Feed Tracking" in the Salesforce.com online help. This field is available in API version 18.0 and later.
`enableHistory`	boolean	Indicates whether the custom object is enabled for audit history (`true`) or not (`false`).
`enableReports`	boolean	Indicates whether the custom object is enabled for reports (`true`) or not (`false`).
`fields`	CustomField[]	Represents one or more fields in the object.
`fullName`	string	Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when

Field Name	Field Type	Description
		creating, updating, or deleting. See `create()` to see an example of this field specified for a call. This value cannot be `null`.
`gender`	Gender	Gender of the name to support translation for languages that indicate gender in nouns. Valid values are: • `Neuter` • `Masculine` • `Feminine`
`household`	boolean	This field supports relationship groups, a feature available only with the Salesforce for Wealth Management. For more information, see “Salesforce for Wealth Management Overview” in the Salesforce.com online help.
`label`	string	Label that represents the object throughout the Salesforce.com user interface.
`listViews`	ListView[]	Represents one or more *list views* associated with the object.
`namedFilter`	NamedFilter[]	Represents the metadata associated with a lookup filter. Use this metadata type to create, update, or delete lookup filter definitions. This field is available in API version 17.0 and later.
`nameField`	CustomField	Required. The field that this object's name is stored in. Every custom object must have a name, usually a string or autonumber. Identifier for the custom object record. This name appears in page layouts, related lists, lookup dialogs, search results, and key lists on tab home pages. By default, this field is added to the custom object page layout as a required field. Every custom object must have a name, usually a string or autonumber.
`pluralLabel`	string	Plural version of the `label` value.
`recordTypes`	RecordType[]	An array of one or more record types defined for this object.
`recordTypeTrackFeedHistory`	boolean	Indicates whether the record type is enabled for feed tracking (`true`) or not (`false`). To set this field to `true`, the `enableFeeds` field on the associated CustomObject must also be `true`. For more information, see “Customizing Chatter Feed Tracking” in the Salesforce.com online help. This field is available in API version 19.0 and later.

Field Name	Field Type	Description
`recordTypeTrackHistory`	boolean	Indicates whether history tracking is enabled for this record type (`true`) or not (`false`). To set `recordTypeTrackHistory` to true the `enableHistory` field on the associated custom object must also be `true`. This field is available in API version 19.0 and later.
`searchLayouts`	SearchLayouts	The *Search Layouts* related list information for a custom object.
`sharingModel`	SharingModel	Indicates the sharing model for this custom object. Valid values are: • `Private` • `Read` • `ReadWrite` **Note:** You can't change the value of this field through the Metadata API; you must use the Web interface.
`sharingReasons`	SharingReason[]	The reasons why the custom object is being shared.
`sharingRecalculations`	SharingRecalculation[]	A list of custom sharing recalculations associated with the custom object.
`startsWith`	StartsWith (enumeration of type string)	Indicates whether the name starts with a vowel, consonant, or is a special character. This is used for languages where words need different treatment depending on the first character. Valid values are listed in StartsWith on page 125.
`validationRules`	ValidationRule[]	An array of one or more validation rules on this object.
`webLinks`	Weblink[]	An array of one or more weblinks defined for the object.

Declarative Metadata Additional Components

CustomObject definitions may include additional components which are defined in the custom object for declarative metadata. The following components are defined in the CustomObject:

- ActionOverride
- BusinessProcess
- CustomField
- ListView
- NamedFilter
- RecordType
- SearchLayouts
- SharingReason
- SharingRecalculation
- ValidationRule
- Weblink

Declarative Metadata Sample Definition

```xml
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <deploymentStatus>Deployed</deploymentStatus>
    <description>just a test object with one field for eclipse ide testing</description>
    <fields>
        <fullName>Comments__c</fullName>
        <description>add your comments about this object here</description>
       <inlineHelpText>This field contains comments made about this object</inlineHelpText>

        <label>Comments</label>
        <length>32000</length>
        <type>LongTextArea</type>
        <visibleLines>30</visibleLines>
    </fields>
    <label>MyFirstObject</label>
    <nameField>
        <label>MyFirstObject Name</label>
        <type>Text</type>
    </nameField>
    <pluralLabel>MyFirstObjects</pluralLabel>
    <sharingModel>ReadWrite</sharingModel>
</CustomObject>
```

See Also:

CustomField
Metadata
Picklist
SearchLayouts
Weblink
CustomObjectTranslation
ListView

ActionOverride

Represents an action override on a custom object. Use it to create, update, edit, or delete action overrides.

Declarative Metadata File Suffix and Directory Location

Action overrides are defined as part of a custom object.

Version

Action overrides are available in API version 18.0 and later.

Fields

Unless otherwise noted, all fields are createable, filterable, and nillable.

Field Name	Field Type	Description
`actionName`	string	Required. The possible values are the same as the actions you can override: • `accept`

Field Name	Field Type	Description
		• `clone` • `delete` • `edit` • `list` • `new` • `tab` • `view`
`comment`	string	Any comments you want associated with the override.
`content`	string	Set this field if `type` is set to `scontrol` or `visualforce`. It refers to the name of the s-control or Visualforce page to use as the override. To reference installed components, use the format of ***`Component_ namespace__Component_name`***.
`type`	ActionOverrideType(enumeration of type string)	Required. Represents the type of action override. Valid values are described in ActionOverrideType.

ActionOverrideType

ActionOverrideType is an enumeration of type string that defines which kind of action override to use. The valid values are:

- `default`—The override uses a custom override provided by an installed package. If there isn't one available, the standard Salesforce.com behavior is used.
- `scontrol`—The override uses behavior from an s-control.
- `standard`—The override uses regular Salesforce.com behavior.
- `visualforce`—The override uses behavior from a Visualforce page.

Declarative Metadata Sample Definitions

You can define an action like this:

```
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <actionOverrides>
        <actionName>edit</actionName>
        <type>visualforce</type>
        <content>myEditVFPage</content>
        <comment>This edit action is a lot safer.</comment>
     </actionOverrides>
</CustomObject>
```

With the previous definition, calling `retrieve()` presents:

```
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <actionOverrides>
        <actionName>edit</actionName>
        <type>default</type>
     </actionOverrides>
</CustomObject>
```

If a subscriber installed a package with the previous metadata, you can override the behavior by editing the XML. For example, if you want the regular Salesforce.com behavior, use:

```
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <actionOverrides>
        <actionName>edit</actionName>
        <type>standard</type>
     </actionOverrides>
</CustomObject>
```

See Also:

CustomObject

BusinessProcess

The BusinessProcess metadata type enables you to display different picklist values for users based on their profile. It extends the Metadata metadata type and inherits its `fullName` field.

Multiple business processes allow you to track separate sales, support, and lead lifecycles. A sales, support, lead, or solution process is assigned to a record type. The record type determines the user profiles that are associated with the business process. For more information, see " Managing Multiple Business Processes " in the Salesforce.com online help.

Declarative Metadata File Suffix and Directory Location

Business processes are defined as part of the custom object or standard object definition. See CustomObject for more information.

Version

BusinessProcess components are available in API version 17.0 and later.

Fields

Field	Field Type	Description
`description`	string	Description for the business process.
`fullName`	string	The name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`isActive`	boolean	Indicates if the business process is active (`true`) or not (`false`).
`namespacePrefix`	string	The namespace of the developer organization where the package was created.
`values`	PicklistValue[]	A list of picklist values associated with this business process.

Declarative Metadata Sample Definition

A sample XML definition of a lead business process is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
....
    <businessProcesses>
        <fullName>HardwareLeadProcess</fullName>
        <description>Lead Process for hardware division</description>
        <isActive>true</isActive>
        <values>
            <fullName>Closed - Converted</fullName>
            <default>false</default>
        </values>
        <values>
            <fullName>CustomLeadStep1</fullName>
            <default>false</default>
        </values>
        <values>
            <fullName>CustomLeadStep2</fullName>
            <default>false</default>
        </values>
        <values>
            <fullName>Open - Not Contacted</fullName>
            <default>false</default>
        </values>
        <values>
            <fullName>Working - Contacted</fullName>
            <default>true</default>
        </values>
    </businessProcesses>
....
</CustomObject>
```

See Also:

CustomObject

CustomField

Represents the metadata associated with a custom field. Use this metadata type to create, update, or delete custom field definitions. It extends the Metadata metadata type and inherits its `fullName` field. You can also use this metadata type to work with customizations of standard picklist fields, such as the `Industry` field for accounts.

You must specify the full name whenever you create or update a custom field. For example, a custom field on a custom object:

```
MyCustomObject__c.MyCustomField__c
```

Another example, a custom field on a standard object:

```
Account.MyAcctCustomField__c
```

Declarative Metadata File Suffix and Directory Location

Custom fields are defined as part of the custom object or standard object definition. See CustomObject for more information.

Note: Retrieving a component of this metadata type in a project makes the component appear in the Profile component as well.

Retrieving Custom Fields on Custom or Standard Objects

When you retrieve a custom or standard object, you return everything associated with the object. However, you can also retrieve only the custom fields for an object by explicitly naming the object and fields in `package.xml`. The following definition in `package.xml` will create the files `objects/MyCustomObject__c.object` and `objects/Account.object__c.object`, each containing one custom field definition.

```
<types>
  <members>MyCustomObject__c.MyCustomField__c</members>
  <members>Account.MyCustomAccountField__c</members>
  <name>CustomField</name>
</types>
```

Version

Custom fields are available in API version 10.0 and later.

Fields

Unless otherwise noted, all fields are createable, filterable, and nillable.

Field Name	Field Type	Description
`caseSensitive`	boolean	Indicates whether the field is case sensitive (`true`) or not (`false`).
`defaultValue`	string	If specified, represents the default value of the field.
`deprecated`	boolean	Reserved for future use.
`description`	string	Description of the field.
`displayFormat`	string	The display format.
`externalId`	boolean	Indicates whether the field is an external ID field (`true`) or not (`false`).
`formula`	string	If specified, represents a formula on the field.
`formulaTreatBlankAs`	TreatBlanksAs (enumeration of type string)	Indicates how to treat blanks in a formula. Valid values are `BlankAsBlank` and `BlankAsZero`.
`fullName`	string	Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call. This value cannot be `null`.
`indexed`	boolean	Indicates if the field is indexed. If this field is unique or the `externalId` is set true, the `isIndexed` value is set to true. This field has been deprecated as of version 14.0 and is only provided for backward compatibility.
`inlineHelpText`	string	Represents the content of field-level help. For more information, see "Defining Field-Level Help" in the Salesforce.com online help.

Field Name	Field Type	Description
`label`	string	Label for the field. You cannot update the label for standard picklist fields, such as the `Industry` field for accounts.
`length`	int	Length of the field.
`maskChar`	EncryptedFieldMaskChar (enumeration of type string)	For encrypted fields, specifies the character to be used as a mask. Valid values are enumerated in EncryptedFieldMaskChar. For more information on encrypted fields, see “About Encrypted Custom Fields” in the Salesforce.com online help.
`maskType`	EncryptedFieldMaskType (enumeration of type string)	For encrypted text fields, specifies the format of the masked and unmasked characters in the field. Valid values are enumerated in EncryptedFieldMaskType For more information on encrypted fields, see “About Encrypted Custom Fields” in the Salesforce.com online help.
`picklist`	Picklist	If specified, the field is a picklist, and this field enumerates the picklist values and labels.
`populateExistingRows`	boolean	Indicates whether existing rows will be populated (`true`) or not (`false`).
`precision`	int	Indicates the precision for number values.
`referenceTo`	string	If specified, indicates a reference this field has to another object.
`relationshipLabel`	string	Label for the relationship.
`relationshipName`	string	If specified, indicates the value for one-to-many relationships. For example, in the object MyObject that had a relationship to YourObject, the relationship name might be YourObjects.
`relationshipOrder`	int	This field is valid for all master-detail relationships, but the value is only non-zero for junction objects. A junction object has two master-detail relationships, and is analogous to an association table in a many-to-many relationship. Junction objects must define one parent object as primary (0), the other as secondary (1). The definition of primary or secondary affects delete behavior and inheritance of look and feel, and record ownership for junction objects. For more information, see the Salesforce.com online help. 0 or 1 are the only valid values, and 0 is always the value for objects that are not junction objects.
`required`	boolean	Indicates whether the field requires a value on creation (`true`) or not (`false`).
`scale`	int	Indicates the scale for the field.
`startingNumber`	int	If specified, indicates the starting number for the field.

Field Name	Field Type	Description
`stripMarkup`	boolean	Set to `true` to remove markup, or `false` to preserve markup. Used when converting a rich text area to a long text area.
`summarizedField`	string	Represents the field on the detail row that is being summarized. This field cannot be null unless the `summaryOperation` value is `count`.
`summaryFilterItems`	FilterItem[]	Represents the set of filter conditions for this field if it is a summary field. This field will be summed on the child if the filter conditions are met.
`summaryForeignKey`	string	Represents the master-detail field on the child that defines the relationship between the parent and the child.
`summaryOperation`	SummaryOperations (enumeration of type string)	Represents the sum operation to be performed. Valid values are enumerated in SummaryOperations.
`trackFeedHistory`	boolean	Indicates whether the field is enabled for feed tracking (`true`) or not (`false`). To set this field to `true`, the `enableFeeds` field on the associated CustomObject must also be `true`. For more information, see "Customizing Chatter Feed Tracking" in the Salesforce.com online help. This field is available in API version 18.0 and later.
`trackHistory`	boolean	Indicates whether history tracking is enabled for the field (`true`) or not (`false`). For more information, see "Tracking Field History" in the Salesforce.com online help.
`trueValueIndexed`	boolean	This is only relevant for a checkbox field. If set, true values are built into the index. This field has been deprecated as of version 14.0 and is only provided for backward compatibility.
`type`	FieldType	Required. Indicates the field type for the field. Valid values are enumerated in FieldType.
`unique`	boolean	Indicates whether the field is unique (`true`) or not (`false`).
`visibleLines`	int	Indicates the number of lines displayed for the field.
`writeRequiresMasterRead`	boolean	Sets the minimum sharing access level required on the master record to create, edit, or delete child records. This field applies only to master-detail or junction object custom field types. • `true` - Allows users with "Read" access to the master record permission to create, edit, or delete child records. This setting makes sharing less restrictive. • `false` - Allows users with "Read/Write" access to the master record permission to create, edit, or delete child records. This setting is more restrictive than `true`, and is the default value.

Field Name	Field Type	Description
		For junction objects, the most restrictive access from the two parents is enforced. For example, if you set to `true` on both master-detail fields, but users have "Read" access to one master record and "Read/Write" access to the other master record, users won't be able to create, edit, or delete child records.

Custom fields use additional data types. For more information, see Metadata Field Types on page 124.

EncryptedFieldMaskChar

This field type is used in `maskChar`. It is a string with two valid values: `asterisk` or `X`. For more information on encrypted fields, see About Encrypted Custom Fields in the Salesforce.com online help.

EncryptedFieldMaskType

This field type is used in `maskType`. Valid values are:

`all`
: All characters in the field are hidden. This option is equivalent to the `Mask All Characters` option in Salesforce.com.

`creditCard`
: The first 12 characters are hidden and the last four display. This option is equivalent to the `Credit Card Number` option in Salesforce.com.

`ssn`
: The first five characters are hidden and the last four display. This option is equivalent to the `Social Security Number` option in Salesforce.com.

`lastFour`
: All characters are hidden but the last four display. This option is equivalent to the `Last Four Characters Clear` option in Salesforce.com.

`sin`
: All characters are hidden but the last four display. This option is equivalent to the `Social Insurance Number` option in Salesforce.com.

`nino`
: All characters are hidden. Salesforce.com automatically inserts spaces after each pair of characters if the field contains nine characters. This option is equivalent to the `National Insurance Number` option in Salesforce.com.

For more information on encrypted fields, see About Encrypted Custom Fields in the Salesforce.com online help.

FilterItem

FilterItem is used in `summaryFilterItems`. It contains the following properties:

Field	Field Type	Description
`field`	string	Represents the field specified in the filter.

Field	Field Type	Description
`operation`	FilterOperation (enumeration of type string)	Represents the filter operation for this filter item. Valid values are enumerated in FilterOperation.
`value`	string	Represents the value of the filter item being operated upon, for example, if the filter is `my_number_field__c > 1`, the value of `value` is `1`.
`valueField`	string	Specifies if the final column in the filter contains a field or a field value

FilterOperation

This is an enumeration of type string that lists different filter operations. Valid values are:

- `equals`
- `notEqual`
- `lessThan`
- `greaterThan`
- `lessOrEqual`
- `greaterOrEqual`
- `contains`
- `notContain`
- `startsWith`
- `includes`
- `excludes`

SummaryOperations

Represents the type of a `summaryOperation`. Valid values are:

- `Count`
- `Min`
- `Max`
- `Sum`

Declarative Metadata Sample Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
....
<fields>
        <fullName>Comments__c</fullName>
        <description>add your comments about this object here</description>
       <inlineHelpText>This field contains comments made about this object</inlineHelpText>

        <label>Comments</label>
        <length>32000</length>
        <type>LongTextArea</type>
        <visibleLines>30</visibleLines>
    </fields>
```

```
....
</CustomObject>
```

See Also:

CustomObject
Picklist
Metadata
NamedFilter

ListView

ListView allows you to see a filtered list of records such as contacts, accounts, or custom objects. It extends the Metadata metadata type and inherits its `fullName` field. See "Creating Custom List Views" in the Salesforce.com online help.

Note: List views with the Visible only to me `Restrict Visibility` option are not accessible in the Metadata API. Each of these list views is associated with a particular user.

Declarative Metadata File Suffix and Directory Location

List views are stored within a CustomObject component. The component can represent a custom object or a standard object, such as an account.

Version

ListView components for custom objects are available in API version 14.0 and later. ListView components for standard objects, such as accounts, are available in API version 17.0 and later.

Fields

Field	Field Type	Description
`booleanFilter`	string	This field represents an Advanced Option for a filter. Advanced Options in filters allow you to build up filtering conditions that use a mixture of AND and OR boolean operators across multiple filter line items. For example, `(1 AND 2) OR 3` finds records that match both the first two filter line items or the third. See "Working with Advanced Filter Conditions" in the Salesforce.com online help.
`columns`	string[]	The list of fields in the list view. The field name relative to the custom object name, for example *`MyCustomField__c`*, is specified for each custom field.
`division`	string	If your organization uses divisions to segment data and you have the "Affected by Divisions" permission, records in the list view must match this division. This field is only available if you are searching all records. This field is available in API version 17.0 and later.
`filterScope`	FilterScope (enumeration of type string)	Required. This field indicates whether you are filtering by owner or viewing all records.

Field	Field Type	Description
`filters`	ListViewFilter[]	The list of filter line items.
`fullName`	string	Required. Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`label`	string	Required. The list view name.
`language`	Language	The language used for filtering if your organization uses the Translation Workbench and you are using the `startsWith` or `contains` operator. The values entered as search terms must be in the same language as the filter language. See "Entering Filter Criteria" in the Salesforce.com online help. For a list of valid language values, see Language. This field is available in API version 17.0 and later.
`queue`	string	The name of a queue. Objects are sometimes assigned to a queue so that the users who have access to the queue can monitor and manage them. When you create a queue, a corresponding list view is automatically created. See "Setting Up Queues" in the Salesforce.com online help.
`sharedTo`	SharedTo	Sharing access for the list view. This field is available in API version 17.0 and later.

ListViewFilter

ListViewFilter represents a filter line item.

Field	Field Type	Description
`filter`	string	Required. Represents the field specified in the filter.
`operation`	FilterOperation (enumeration of type string)	Required. The operation used by the filter, such as `equals`. The valid values are listed in FilterOperation.
`value`	string	Represents the value of the filter item being operated upon, for example, if the filter is `my_number_field__c > 1`, the value of `value` is `1`.

FilterScope

This is an enumeration of type string that represents the filtering criteria for the records. The valid values are listed in the table below:

Enumeration Value	Description
`Everything`	All records, for example All Opportunities.
`Mine`	Records owned by the user running the list view, for example My Opportunities.

Enumeration Value	Description
`Queue`	Records assigned to a queue.
`Delegated`	Records delegated to another user for action: for example, a delegated task. This option is available in API version 17.0 and later.
`MyTerritory`	Records in the territory of the user seeing the list view. This option is available if territory management is enabled for your organization. This option is available in API version 17.0 and later.
`MyTeamTerritory`	Records in the territory of the team of the user seeing the list view. This option is available if territory management is enabled for your organization. This option is available in API version 17.0 and later.
`Team`	Records assigned to a team. This option is available in API version 17.0 and later.

SharedTo

SharedTo defines the sharing access for a list view or a folder. See "Sharing Considerations" and "About Groups" in the Salesforce.com online help.

Field	Field Type	Description
`groups`	string[]	A list of groups with sharing access.
`roles`	string[]	A list of roles with sharing access.
`rolesAndSubordinates`	string[]	A list of roles with sharing access. All roles below each of these roles in the role hierarchy also have sharing access.
`territories`	string[]	A list of territories with sharing access.
`territoriesAndSubordinates`	string[]	A list of territories with sharing access. All territories below each of these territories in the territory hierarchy also have sharing access.

Declarative Metadata Sample Definition

A sample XML definition of a list view in a custom object is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
. . .
    <listViews>
        <fullName>All_Mileages</fullName>
        <filterScope>all</filterScope>
        <label>All Mileages</label>
    </listViews>
    <listViews>
        <fullName>My_Mileages</fullName>
        <booleanFilter>1 AND 2</booleanFilter>
        <columns>NAME</columns>
        <columns>CREATED_DATE</columns>
        <filterScope>mine</filterScope>
        <filters>
            <field>NAME</field>
            <operation>equals</operation>
            <value>Eric Bristow</value>
        </filters>
        <filters>
```

```
            <field>City__c</field>
            <operation>equals</operation>
            <value>Paris</value>
        </filters>
        <label>My Mileages</label>
    </listViews>
. . .
</CustomObject>
```

See Also:

CustomObject
Sample package.xml Manifest Files

NamedFilter

Represents the metadata associated with a lookup filter. Use this metadata type to create, update, or delete lookup filter definitions. It extends the Metadata metadata type and inherits its `fullName` field. You can also use this metadata type to work with customizations of lookup filters on standard fields.

Note: The namedFilter appears as a child of the target object of the associated lookup field.

Declarative Metadata File Suffix and Directory Location

Lookup filters are defined as part of the custom object or standard object definition. See CustomObject for more information.

Note: Retrieving a component of this metadata type in a project makes the component appear in the Profile component as well.

Version

Lookup filters are available in API version 17.0 and later.

Fields

Unless otherwise noted, all fields are createable, filterable, and nillable.

Field Name	Field Type	Description
`active`	boolean	Required. Indicates whether or not the lookup filter is active.
`booleanFilter`	string	Specifies advanced filter conditions. For more information on advanced filter conditions, see "Working with Advanced Filter Conditions" in the Salesforce.com online help.
`description`	string	A description of what this filter does.
`errorMessage`	string	The error message that appears if the lookup filter fails.

Field Name	Field Type	Description
`field`	string	Required. The `fullName` of the custom or standard field associated with the lookup filter. You can associate one relationship field with each lookup filter, and vice-versa. **Note:** You cannot update a field associated with a lookup filter.
`filterItems`	FilterItems[]	Required. The set of filter conditions.
`infoMessage`	string	The information message displayed on the page. Use to describe things the user might not understand, such as why certain items are excluded in the lookup filter.
`fullName`	string	Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call. This value cannot be `null`.
`isOptional`	boolean	Required. Indicates whether or not the lookup filter is optional.
`name`	string	Required. The name of the lookup filter. If you create this field in the user interface, a name is automatically assigned. If you create this field through the Metadata API, you must include the `name` field.
`sourceObject`	string	The object that contains the lookup field that uses this lookup filter. Set this field if the lookup filter references fields on the source object.

Lookup filters use additional data types. For more information, see Metadata Field Types on page 124.

FilterItems

FilterItems contains the following properties:

Field	Field Type	Description
`field`	string	Represents the field specified in the filter.
`operation`	FilterOperation (enumeration of type string)	Represents the filter operation for this filter item. Valid values are enumerated in FilterOperation.
`value`	string	Represents the value of the filter item being operated upon, for example, if the filter is `my_number_field__c > 1`, the value of `value` is `1`.

FilterOperation

This is an enumeration of type string that lists different filter operations. Valid values are:

- `equals`
- `notEqual`
- `lessThan`
- `greaterThan`
- `lessOrEqual`
- `greaterOrEqual`
- `contains`
- `notContain`
- `startsWith`
- `includes`
- `excludes`

Declarative Metadata Sample Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
....
    <namedfilters>
        <fullName>nf_Acc</fullName>
        <active>true</active>
        <booleanFilter>1 OR 2</booleanFilter>
        <field>Account.lk__c</field>
        <filterItems>
            <field>Account.Phone</field>
            <operation>notEqual</operation>
            <value>x</value>
        </filterItems>
        <filterItems>
            <field>Account.Fax</field>
            <operation>notEqual</operation>
            <value>y</value>
        </filterItems>
        <name>Acc</name>
        <sourceObject>Account</sourceObject>
    </namedfilters>
....
</CustomObject>
```

See Also:

CustomObject
Picklist
Metadata
CustomField

Picklist

Represents a picklist definition for a custom field in a custom object or a custom or standard field in a standard object, such as an account. Note that picklist values cannot be deleted from a picklist that has been saved to your organization, since data rows might exist that would need to be interactively remapped.

Version

Picklists for custom fields in custom objects are available in API version 12.0 and later. Picklists for custom or standard fields in standard objects, such as accounts, are available in API version 16.0 and later.

Declarative Metadata File Suffix and Directory Location

Picklist definitions are included in the custom object and field with which they are associated.

Fields

Picklist contains the following fields:

Field Name	Field Type	Description
controllingField	string	The fullName of the controlling field if this is a dependent picklist. A dependent picklist works in conjunction with a controlling picklist or checkbox to filter the available options. The value chosen in the controlling field affects the values available in the dependent field. This field is available in API version 14.0 and later.
picklistValues	PicklistValue[]	Required. Represents a set of values for a picklist.
sorted	boolean	Required. Indicates whether values should be sorted (true), or not (false).

PicklistValue

This metadata type defines a value in the picklist and specifies whether this value is the default value. It extends the Metadata metadata type and inherits its fullName field. Note the following when working with picklist values for standard objects:

- When you retrieve a standard object, you all picklist values are retrieved, not just the customized picklist values.
- When you deploy changes to standard picklist fields, you cannot delete existing picklist values.

Field Name	Field Type	Description
allowEmail	boolean	Indicates whether this value lets users email a quote PDF (true), or not (false). This field is only relevant for the Status field in quotes. This field is available in API version 18.0 and later.
closed	boolean	Indicates whether this value is associated with a closed status (true), or not (false). This field is only relevant for the standard Status field in cases and tasks. This field is available in API version 16.0 and later.
color	string	Indicates the color assigned to the picklist value when used in charts on reports and dashboards. The color is in hexadecimal format; for example #FF6600. If a color is not specified, it will be assigned dynamically on chart generation. This field is available in API version 17.0 and later.
controllingFieldValues	string[]	A list of values in the controlling field that are linked to this picklist value. The controlling field can be a checkbox or a picklist. This field is available in API version 14.0 and later. The values in the list depend on the field type: • Checkbox: checked or unchecked. • Picklist: The fullName of the picklist value in the controlling field.

Field Name	Field Type	Description
`converted`	boolean	Indicates whether this value is associated with a converted status (`true`), or not (`false`). This field is only relevant for the standard `Lead Status` field in leads. Your organization can set its own guidelines for determining when a lead is qualified, but typically, you will want to convert a lead as soon as it becomes a real opportunity that you want to forecast. For more information, see "Converting Leads" in the Salesforce.com online help. This field is available in API version 16.0 and later.
`cssExposed`	boolean	Indicates whether this value is available in your Self-Service Portal (`true`), or not (`false`). This field is only relevant for the standard `Case Reason` field in cases. Self-Service provides an online support channel for your customers - allowing them to resolve their inquiries without contacting a customer service representative. For more information about Self-Service, see "Setting Up Self-Service" in the Salesforce.com online help. This field is available in API version 16.0 and later.
`default`	boolean	Required. Indicates whether this value is the default picklist value in the specified picklist (`true`), or not (`false`).
`description`	string	Description of a custom picklist value. This field is only relevant for the standard `Stage` field in opportunities. It is useful to include a description for a customized picklist value so that the historical reason for creating it can be tracked. This field is available in API version 16.0 and later.
`forecastCategory`	ForecastCategories (enumeration of type string)	Indicates whether this value is associated with a forecast category (`true`), or not (`false`). This field is only relevant for the standard `Stage` field in opportunities. For more information about forecast categories, including the valid string values listed below, see " Working with Forecast Categories " in the Salesforce.com online help. • Omitted • Pipeline • BestCase • Forecast • Closed This field is available in API version 16.0 and later.
`fullName`	string	The name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`highPriority`	boolean	Indicates whether this value is a high priority item (`true`), or not (`false`). This field is only relevant for the standard `Priority` field in tasks. For more information about tasks, see "Creating Tasks" in the Salesforce.com online help. This field is available in API version 16.0 and later.

Field Name	Field Type	Description
probability	int	Indicates whether this value is a probability percentage (`true`), or not (`false`). This field is only relevant for the standard `Stage` field in opportunities. For more information about opportunities, see "Opportunities Overview" in the Salesforce.com online help. This field is available in API version 16.0 and later.
reverseRole	string	A picklist value corresponding to a reverse role name for a partner. If the role is "subcontractor", then the reverse role might be "general contractor". Assigning a partner role to an account in Salesforce.com creates a reverse partner relationship so that both accounts list the other as a partner. This field is only relevant for partner roles. For more information, see "Partner Fields" in the Salesforce.com online help. This field is available in API version 18.0 and later.
reviewed	boolean	Indicates whether this value is associated with a reviewed status (`true`), or not (`false`). This field is only relevant for the standard `Status` field in solutions. For more information about opportunities, see "Creating Solutions" in the Salesforce.com online help. This field is available in API version 16.0 and later.
won	boolean	Indicates whether this value is associated with a closed or won status (`true`), or not (`false`). This field is only relevant for the standard `Stage` field in opportunities. This field is available in API version 16.0 and later.

Java Sample

The following sample uses a picklist. For a complete sample of using a picklist with record types and profiles, see Profile on page 174.

```
private void setPicklistValues()
    throws Exception
{
    // Create a picklist
    Picklist expenseStatus = new Picklist();
    PicklistValue unsubmitted = new PicklistValue();
    unsubmitted.setFullName("Unsubmitted");
    PicklistValue submitted = new PicklistValue();
    submitted.setFullName("Submitted");
    PicklistValue approved = new PicklistValue();
    approved.setFullName("Approved");
    PicklistValue rejected = new PicklistValue();
    rejected.setFullName("Rejected");
    expenseStatus.setPicklistValues(new PicklistValue[]
      {unsubmitted, submitted, approved, rejected});

    CustomField expenseStatusField = new CustomField();
    expenseStatusField.setFullName("ExpenseReport__c.ExpenseStatus__c");
    expenseStatusField.setLabel("Expense Report Status");
    expenseStatusField.setType(FieldType.Picklist);
    expenseStatusField.setPicklist(expenseStatus);
    AsyncResult[] ars =
        metadataBinding.create(expenseStatusField);
}
```

Declarative Metadata Sample Definition

The following sample shows usage for picklists, including dependent picklists, in a custom object. The `isAmerican__c` checkbox controls the list of manufacturers shown in the `manufacturer__c` picklist. The `manufacturer__c` checkbox in turn controls the list of models shown in the `model__c` picklist.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <deploymentStatus>Deployed</deploymentStatus>
    <enableActivities>true</enableActivities>
    <fields>
        <fullName>isAmerican__c</fullName>
        <defaultValue>false</defaultValue>
        <label>American Only</label>
        <type>Checkbox</type>
    </fields>
    <fields>
        <fullName>manufacturer__c</fullName>
        <label>Manufacturer</label>
        <picklist>
            <controllingField>isAmerican__c</controllingField>
            <picklistValues>
                <fullName>Chrysler</fullName>
                <controllingFieldValues>checked</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
                <fullName>Ford</fullName>
                <controllingFieldValues>checked</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
                <fullName>Honda</fullName>
                <controllingFieldValues>unchecked</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
                <fullName>Toyota</fullName>
                <controllingFieldValues>unchecked</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <sorted>false</sorted>
        </picklist>
        <type>Picklist</type>
    </fields>
    <fields>
        <fullName>model__c</fullName>
        <label>Model</label>
        <picklist>
            <controllingField>manufacturer__c</controllingField>
            <picklistValues>
                <fullName>Mustang</fullName>
                <controllingFieldValues>Ford</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
                <fullName>Taurus</fullName>
                <controllingFieldValues>Ford</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
                <fullName>PT Cruiser</fullName>
                <controllingFieldValues>Chrysler</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
```

```
                <fullName>Pacifica</fullName>
                <controllingFieldValues>Chrysler</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
                <fullName>Accord</fullName>
                <controllingFieldValues>Honda</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
                <fullName>Civic</fullName>
                <controllingFieldValues>Honda</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
                <fullName>Prius</fullName>
                <controllingFieldValues>Toyota</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <picklistValues>
                <fullName>Camry</fullName>
                <controllingFieldValues>Toyota</controllingFieldValues>
                <default>false</default>
            </picklistValues>
            <sorted>false</sorted>
        </picklist>
        <type>Picklist</type>
    </fields>
....
</CustomObject>
```

The following sample shows usage for the standard `Stage` field in opportunities.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <fields>
        <fullName>StageName</fullName>
        <picklist>
            <picklistValues>
                <fullName>Prospecting</fullName>
                <default>false</default>
                <forecastCategory>Pipeline</forecastCategory>
                <probability>10</probability>
            </picklistValues>
            <picklistValues>
                <fullName>Qualification</fullName>
                <default>false</default>
                <forecastCategory>Pipeline</forecastCategory>
                <probability>10</probability>
            </picklistValues>
            <picklistValues>
                <fullName>Needs Analysis</fullName>
                <default>false</default>
                <forecastCategory>Pipeline</forecastCategory>
                <probability>20</probability>
            </picklistValues>
            ...
        </picklist>
    </fields>
<CustomObject>
```

RecordType

Represents the metadata associated with a record type. Record types allow you to offer different business processes, picklist values, and page layouts to different users based on their profiles. For more information, see "Managing Record Types" in the Salesforce.com online help. Use this metadata type to create, update, or delete record type definitions for a custom object. It extends the Metadata metadata type and inherits its `fullName` field.

Note: Retrieving a component of this metadata type in a project makes the component appear in the Profile component as well.

Version

Record types are available in API version 12.0 and later.

Fields

Field	Field Type	Description
`active`	boolean	Required. Indicates whether or not the record type is active.
`businessProcess`	string	The `fullName` of the business process associated with the record type. This field is required in record types for lead, opportunity, solution, and case, and not allowed otherwise. See BusinessProcess on page 96. This field is available in API version 17.0 and later.
`description`	string	Record type description. Maximum of 255 characters.
`fullName`	string	Record type name. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. If this field contained characters before version 14.0 that are no longer allowed, the characters were stripped out of this field, and the previous value of the field was saved in the `label` field. Inherited from the Metadata component, this field is not defined in the WSDL for this component. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call. This value cannot be `null`.
`label`	string	Required. Descriptive label for the record type. The list of characters allowed in the `fullName` field has been reduced for versions 14.0 and later. This field contains the value contained in the `fullName` field before version 14.0.
`picklistValues`	RecordTypePicklistValue[]	Represents a set of values for a picklist.

RecordTypePicklistValue

RecordTypePicklistValue represents the combination of picklists and valid values that define a record type:

Field Name	Field Type	Description
picklist	string	Required. The name of the picklist.
values	PicklistValue	One or more of the picklist values in the picklist. Each value defined is available in the record type that contains this component.

Java Sample

The following sample uses two record types. For the complete sample the includes profiles and picklists, see Profile on page 174.

```
private void recordTypeSample()
    throws Exception
{
    // Employees and managers have different access
    // to the state of the expense sheet
    RecordType edit = new RecordType();
    edit.setFullName("ExpenseReport__c.Edit");
    RecordTypePicklistValue editStatuses = new RecordTypePicklistValue();
    editStatuses.setPicklist("ExpenseStatus__c");
    editStatuses.setValues(new PicklistValue[] {unsubmitted, submitted});
    edit.setPicklistValues(new RecordTypePicklistValue[] {editStatuses});
    AsyncResult[] arsEdit =
        metadataBinding.create(edit);

    RecordType approve = new RecordType();
    approve.setFullName("ExpenseReport__c.Approve");
    RecordTypePicklistValue approveStatuses = new RecordTypePicklistValue();
    approveStatuses.setPicklist("ExpenseStatus__c");
    approveStatuses.setValues(new PicklistValue[] {approved, rejected});
    approve.setPicklistValues(new RecordTypePicklistValue[] {approveStatuses});
    AsyncResult[] arsApprove =
        metadataBinding.create(approve);
}
```

Declarative Metadata Sample Definition

The definition of a record type in a custom object is shown below:

```
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
. . .
   <recordTypes>
        <fullName>My First Recordtype</fullName>
    </recordTypes>
 . . .
</CustomObject>
```

SearchLayouts

Represents the metadata associated with the Search Layouts related list for a custom object. You can customize which custom object fields display for users in search results, in lookup dialogs, and in the key lists on custom tab home pages. For more information, see “Customizing Search Layouts for Custom Objects” in the Salesforce.com online help.

Version

Search layouts are available in API version 14.0 and later.

Fields

Field	Field Type	Description
`customTabListAdditionalFields`	string[]	The list of fields displayed in the Recent *`Custom Object Name`* list view on a custom tab associated with the custom object. The `name` field is mandatory and is always displayed as the first column header, so it is not included in this list; all additional fields are included. The field name relative to the custom object name, for example *`MyCustomField__c`*, is specified for each custom field.
`excludedStandardButtons`	string[]	The list of standard buttons excluded from the search layout.
`listViewButtons`	string[]	The list of buttons available in list views for the custom object. This field is equivalent to the Buttons Displayed value in the *`Custom Object Name`* `List View` in the Search Layouts related list on the custom object detail page in the Salesforce.com user interface. For more information, see "Lookup Dialog Search" in the Salesforce.com online help.
`lookupDialogsAdditionalFields`	string[]	The list of fields displayed in a lookup dialog for the custom object. The `name` field is mandatory and is always displayed as the first column header, so it is not included in this list; all additional fields are included. The field name relative to the custom object name, for example *`MyCustomField__c`*, is specified for each custom field. Salesforce.com objects often include one or more *lookup fields* that allow users to associate two records together in a relationship. For example, a contact record includes an `Account` lookup field that represents the relationship between the contact and the organization with which the contact is associated. A lookup search dialog helps you search for the record associated with the one being edited. Lookup filter fields allow you to filter your lookup search by a customized list of fields in the custom object. This field is equivalent to the `Lookup Dialogs` in the Search Layouts related list on the custom object detail page in the application user interface. For more information, see "Lookup Dialog Search" in the Salesforce.com online help.
`lookupFilterFields`	string[]	The list of fields that can be used to filter enhanced lookups for the custom object. Enhanced lookups are optionally enabled by your administrator. The field name relative to the custom object name, for example *`MyCustomField__c`*, is specified for each custom field. This field is equivalent to the `Lookup Filter Fields` in the Search Layouts related list on the custom object

Field	Field Type	Description
		detail page in the application user interface. For more information, see "Lookup Dialog Search" in the Salesforce.com online help.
`lookupPhoneDialogsAdditionalFields`	string[]	The list of phone-related fields displayed in a lookup dialog for the custom object. The `name` field is mandatory and is always displayed as the first column header, so it is not included in this list; all additional fields are included. The field name relative to the custom object name, for example *`MyCustomField__c`*, is specified for each custom field. This list enables integration of the fields with a SoftPhone dial pad. For more information, see "About CTI 1.0 and 2.0 SoftPhones" in the Salesforce.com online help. This field is equivalent to the `Lookup Phone Dialogs` in the Search Layouts related list on the custom object detail page in the application user interface.
`searchFilterFields`	string[]	The list of fields that can be used to filter a search for the custom object. The field name relative to the custom object name, for example *`MyCustomField__c`*, is specified for each custom field. This field is equivalent to the `Search Filter Fields` in the Search Layouts related list on the custom object detail page in the application user interface.
`searchResultsAdditionalFields`	string[]	The list of fields displayed in a search result for the custom object. The `name` field is mandatory and is always displayed as the first column header, so it is not included in this list; all additional fields are included. The field name relative to the custom object name, for example *`MyCustomField__c`*, is specified for each custom field. This field is equivalent to the `Search Results` in the Search Layouts related list on the custom object detail page in the application user interface.
`searchResultsCustomButtons`	string[]	The list of custom buttons available in a search result for the custom object. The actions associated with the buttons can be applied to any of the records returned in the search result.

Declarative Metadata Sample Definition

A sample definition of search layouts in a custom object is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
. . .
    <searchLayouts>
        <listViewButtons>New</listViewButtons>
        <listViewButtons>Accept</listViewButtons>
        <listViewButtons>ChangeOwner</listViewButtons>
        <lookupDialogsAdditionalFields>firstQuote__c</lookupDialogsAdditionalFields>
        <lookupDialogsAdditionalFields>finalQuote__c</lookupDialogsAdditionalFields>
        <searchResultsAdditionalFields>CREATEDBY_USER</searchResultsAdditionalFields>
     </searchLayouts>
. . .
</CustomObject>
```

See Also:

CustomObject

SharingReason

Apex managed sharing allows developers to use Apex to programmatically share custom objects. When you use Apex managed sharing to share a custom object, only users with the “Modify All Data” permission can add or change the sharing on the custom object's record, and the sharing access is maintained across record owner changes. A sharing reason is used to indicate why sharing was implemented for a custom object. For more information, see “Managing the Sharing Settings” in the Salesforce.com online help.

Use SharingReason to create, update, or delete sharing reason definitions for a custom object. It extends the Metadata metadata type and inherits its `fullName` field.

Version

Sharing reasons are available in API version 14.0 and later.

Fields

Field	Field Type	Description
`fullName`	string	Required. Sharing reason name. The __c suffix is appended to custom sharing reasons. Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`label`	string	Required. Descriptive label for the sharing reason. Maximum of 40 characters.

Declarative Metadata Sample Definition

The definition of a sharing reason in a custom object:

```
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
. . .
    <sharingReasons>
        <fullName>recruiter__c</fullName>
        <label>Recruiter</label>
    </sharingReasons>
 . . .
</CustomObject>
```

SharingRecalculation

Developers can write Apex classes that recalculate the Apex managed sharing for a specific custom object. For more information, see “Recalculating Apex Managed Sharing” in the Salesforce.com online help.

Version

Sharing recalculations are available in API version 14.0 and later.

Fields

Field	Field Type	Description
`className`	string	Required. The Apex class that recalculates the Apex sharing for a custom object. This class must implement the `Database.Batchable` interface.

Declarative Metadata Sample Definition

The definition of a sharing recalculation in a custom object:

```
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
. . .
    <sharingRecalculations>
        <className>RecruiterRecalculation</className>
    </sharingRecalculations>
 . . .
</CustomObject>
```

ValidationRule

Represents a validation rule, which is used to verify that the data a user enters in a record is valid and can be saved. A validation rule contains a formula or expression that evaluates the data in one or more fields and returns a value of `true` or `false`. Validation rules also include an error message that your client application can display to the user when the rule returns a value of `true` due to invalid data. It extends the Metadata metadata type and inherits its `fullName` field.

As of API version 20.0, validation rules can't have compound fields. Examples of compound fields include addresses, first and last names, dependent picklists, and dependent lookups..

Version

Validation rules are available in API version 12.0 and later.

Fields

Field Name	Field Type	Description
`active`	boolean	Required. Indicates whether this validation rule is active, (`true`), or not active (`false`).
`description`	string	A description of the validation rule.

Field Name	Field Type	Description
errorConditionFormula	string	Required. The validation formula, as documented in the validation formula page. See "Defining Validation Rules" in the Salesforce.com online help.
errorDisplayField	string	The fully specified name of a field in the application. If a value is supplied in this field, the value in errorMessage appears next to the specified field. If you do not specify a value, the error message appears at the top of the page.
errorMessage	string	Required. The message that appears if the validation rule fails. The message must be 255 characters or less.
fullName	string	The internal name of the validation rule, with white spaces and special characters escaped for validity. The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. Inherited from the Metadata component, this field is not defined in the WSDL for this component. It must be specified when creating, updating, or deleting. See create() to see an example of this field specified for a call.

Declarative Metadata Sample Definition

A sample XML definition of a validation rule in a custom object is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <deploymentStatus>Deployed</deploymentStatus>
    <fields>
        <fullName>Mommy_Cat__c</fullName>
        <label>Mommy Cat</label>
        <referenceTo>Cat__c</referenceTo>
        <relationshipName>Cats</relationshipName>
        <type>Lookup</type>
    </fields>
    <label>Cat</label>
    <nameField>
        <label>Cat Name</label>
        <type>Text</type>
    </nameField>
    <pluralLabel>Cats</pluralLabel>
    <sharingModel>ReadWrite</sharingModel>
    <validationRules>
        <fullName>CatsRule</fullName>
        <active>true</active>
        <errorConditionFormula>OR(Name = &apos;Milo&apos;,Name =
&apos;Moop&apos;)</errorConditionFormula>
        <validationMessage>Name must be that of one of my cats</validationMessage>
    </validationRules>
</CustomObject>
```

Weblink

Represents a Weblink defined in a custom object. It extends the Metadata metadata type and inherits its fullName field.

Version

Weblinks are available in API version 12.0 and later.

Fields

The Weblink definition contains the following fields.

Field Name	Field Type	Description
`availability`	WebLinkAvailability (enumeration of type string)	Required. Indicates whether the Weblink is only available online (`online`, or if it is also available offline (`offline`).
`description`	string	A description of the Weblink.
`displayType`	WebLinkDisplayType (enumeration of type string)	Represents how this Weblink is rendered. Valid values: • `link` for a hyperlink • `button` for a button • `massAction` for a button attached to a related list
`fullName`	string	The name of the weblink with white spaces and special characters escaped for validity. The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. Inherited from the Metadata component, this field is not defined in the WSDL for this component. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`hasMenubar`	boolean	If the `openType` is `newWindow`, whether to show the browser menu bar for the window (`true` or not (`false`). Otherwise this field should not be specified.
`hasScrollbars`	boolean	If the `openType` is `newWindow`, whether to show the scroll bars for the window (`true`) or not (`false`). Otherwise this field should not be specified.
`hasToolbar`	boolean	If the `openType` is `newWindow`, whether to show the browser toolbar for the window (`true`) or not (`false`). Otherwise this field should not be specified.
`height`	int	Height in pixels of the window opened by this Weblink. Required if the `openType` is `newWindow`, otherwise should not be specified
`isResizable`	boolean	If the `openType` is `newWindow`, whether to allow resizing of the window (`true`) or not (`false`). Otherwise this field should not be specified.
`linkType`	WebLinkType (enumeration of type string)	Required. Represents whether the content of this Weblink is specified by a URL, an sControl, a JavaScript code block, or a Visualforce page. Valid values: • `url`

Field Name	Field Type	Description
		• `sControl` • `javascript` • `page`
`masterLabel`	string	The master label for the Weblink.
`openType`	WebLinkWindowType (enumeration of type string)	Required. When this button is clicked, specifies the window style that will be used to display the content. Valid values: • `newWindow` • `sidebar` • `noSidebar` • `replace` • `onClickJavaScript`
`page`	string	If the value of `linkType` is `page`, this field represents the Visualforce page; otherwise, this field should not be specified.
`position`	WebLinkPosition (enumeration of type string)	If the `openType` is `newWindow`, how the new window should be displayed. Otherwise this field should not be specified. Valid values: • `fullScreen` • `none` • `topLeft`
`protected`	boolean	Required. Indicates whether this sub-component is protected (`true`) or not (`false`). Protected sub-components cannot be linked to or referenced by components or sub-components created in the installing organization.
`requireRowSelection`	boolean	If the `openType` is `massAction`, whether to require individual row selection to execute the action for this button (`true`) or not (`false`). Otherwise this field should not be specified.
`scontrol`	string	If the value of `linkType` is `sControl`, this field represents the name of the sControl; otherwise, this field should not be specified.
`showsLocation`	boolean	If the `openType` is `newWindow`, whether or not to show the browser location bar for the window; otherwise this field should not be specified.
`showsStatus`	boolean	If the `openType` is `newWindow`, whether or not to show the browser status bar for the window. Otherwise, this field should not be specified.
`url`	string	If the value of `linkType` is `url`, this is the URL value. If the value of `linkType` is `javascript`, this is the JavaScript content. If the value neither of these, the this field should not be specified. Content must be escaped in a manner consistent with XML parsing rules.

Field Name	Field Type	Description
`width`	int	Width in pixels of the window opened by this Weblink. Required if the `openType` is `newWindow`, otherwise should not be specified.

Java Sample

The following Java sample shows sample values for Weblink fields:

```
private void webLinkSample()
{
    Weblink weblink = new Weblink();
    weblink.setFullName("googleButton");
    weblink.setUrl("http://www.google.com");
    weblink.setAvailability(WebLinkAvailability.online);
    weblink.setLinktype(WebLinkType.url);
    weblink.setOpentype(WebLinkWindowType.newWindow);
    weblink.setHeight(600);
    weblink.setWidth(600);
    weblink.setShowsLocation(false);
    weblink.setHasScrollbars(true);
    weblink.setHasToolbar(false);
    weblink.setHasMenubar(false);
    weblink.setShowsStatus(false);
    weblink.setIsResizable(true);
    weblink.setPosition(WebLinkPosition.none);
    weblink.setMasterLabel("google");
    weblink.setDisplayType(WebLinkDisplayType.link);
    weblink.setRequireRowSelection(true);
}
```

Declarative Metadata Sample Definition

The following is the definition of a Weblink in a custom object. For related samples, see HomePageComponent on page 161 and HomePageLayout on page 162.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
....
    <webLinks>
        <fullName>detailPageButton</fullName>
        <availability>online</availability>
        <displayType>button</displayType>
        <hasScrollbars>true</hasScrollbars>
        <height>600</height>
        <isResizable>true</isResizable>
        <linkType>url</linkType>
        <masterLabel>detailPageButon</masterLabel>
        <openType>newWindow</openType>
        <position>none</position>
        <protected>false</protected>
        <url>http://google.com</url>
    </webLinks>
    ....
</CustomObject>
```

See Also:

HomePageComponent
HomePageLayout
CustomPageWebLink

Metadata Field Types

These field types extend the field types described in the *Web Services API Developer's Guide*.

Field Type	Objects	What the Field Contains
CustomField	Custom object Custom field	Represents a custom field.
DeploymentStatus	Custom object Custom field	A string which represents the deployment status of a custom object or field. Valid values are: • `InDevelopment` • `Deployed`
FieldType	Custom field	Indicates the type of a custom field. Valid values are: • `AutoNumber` • `Lookup` • `MasterDetail` • `Checkbox` • `Currency` • `Date` • `DateTime` • `Email` • `EncryptedText` • `Number` • `Percent` • `Phone` • `Picklist` • `MultiselectPicklist` • `Summary` • `Text` • `TextArea` • `LongTextArea` • `Summary` • `Url` • `Hierarchy` • `File` • `CustomDataType` • `Html`

Field Type	Objects	What the Field Contains
Gender	Custom object	Gender of the name to support translation for languages that indicate gender in nouns. Valid values are: • `Neuter` • `Masculine` • `Feminine`
Picklist	Custom field	Represents a picklist, a set of labels and values that can be selected from a picklist.
SharingModel	Custom object Custom field	Represents the sharing model for the custom object or custom field. Valid values are: • `Private` • `Read` • `ReadWrite`
StartsWith	Custom object Custom field	Indicates whether the name starts with a vowel, consonant, or is a special character. This is used for languages where words need different treatment depending on the first character. Valid values are: • `Consonant` • `Vowel` • `Special`
TreatBlanksAs	Custom field	Indicates how blanks should be treated. Valid values are: • `BlankAsBlank` • `BlankAsZero`

CustomObjectTranslation

This metadata type allows you to translate custom objects for a variety of languages. It extends the Metadata metadata type and inherits its `fullName` field. The ability to translate component labels is part of the Translation Workbench. For more information, see "Setting Up the Translation Workbench" in the Salesforce.com online help.

Declarative Metadata File Suffix and Directory Location

Translations are stored in a file with a format of *`customObjectName__c-lang`*`.objectTranslation`, where *`customObjectName__c`* is the custom object name, and *`lang`* is the translation language. A sample file name for German translations is `myCustomObject__c-de.objectTranslation`.

Custom object translations are stored in the `objectTranslations` folder in the corresponding package directory.

Version

CustomObjectTranslation components are available in API version 14.0 and later.

Fields

Field	Field Type	Description
`caseValues`	ObjectNameCaseValue[]	Different combinations of the custom object with the definite and indefinite articles, and for the singular and plural cases.
`fields`	CustomFieldTranslation[]	A list of translations for the custom fields associated with the custom object.
`fullName`	string	The name of the custom object and the translation language with a format of *`customObjectName-lang`*, where *`customObjectName`* is the custom object name, and *`lang`* is the translation language. Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`gender`	Gender	Gender of the name to support translation for languages that indicate gender in nouns. Valid values are: • `Neuter` • `Masculine` • `Feminine`
`layouts`	LayoutTranslation[]	A list of page layout translations.
`nameFieldLabel`	string	The label for the name field. Maximum of 765 characters.
`namedFilters`	NamedFilterTranslation[]	A list of translations for lookup filter error messages associated with the custom object.
`recordTypes`	RecordTypeTranslation[]	A list of record type translations.
`sharingReasons`	SharingReasonTranslation[]	A list of sharing reason translations.
`startsWith`	StartsWith (enumeration of type string)	Indicates whether the name starts with a vowel, consonant, or is a special character. This is used for languages where words need different treatment depending on the first character. Valid values are listed in StartsWith on page 125.
`validationRules`	ValidationRuleTranslation[]	A list of validation rule translations.
`webLinks`	WebLinkTranslation[]	A list of web link translations.
`workflowTasks`	WorkflowTaskTranslation[]	A list of workflow task translations.

CustomFieldTranslation

CustomFieldTranslation contains details for a custom field translation. For more details, see CustomField on page 97.

Field	Field Type	Description
`description`	string	Translation for the custom field description.

Field	Field Type	Description
`help`	string	Translation for the text that displays in the field-level help hover text for this field.
`label`	string	Translation for the label. Maximum of 765 characters.
`name`	string	Required. The name of the field relative to the custom object; for example, `MyField__c`.
`picklistValues`	PicklistValueTranslation[]	List of translations for picklist values. See PicklistValue on page 109 for more details.
`relationshipLabel`	string	Translation for a lookup relationship label. A lookup relationship allows a field to be associated with another field. The relationship field allows users to select an option from a list of values defined by the other field. Maximum of 765 characters.

LayoutTranslation

LayoutTranslation contains details for a page layout translation. For more details, see Fields on page 163.

Field	Field Type	Description
`layout`	string	Required. The layout name.
`sections`	LayoutSectionTranslation[]	An array of layout section translations.

LayoutSectionTranslation

LayoutSectionTranslation contains details for a page layout section translation. For more details, see LayoutSection on page 164.

Field	Field Type	Description
`label`	string	Required. Translation for the label. Maximum of 765 characters.
`section`	string	Required. The section name.

NamedFilterTranslation

NamedFilterTranslation shows a list of translations for lookup filter error messages associated with the custom object. See NamedFilter on page 106 for more information.

Field	Field Type	Description
`errorMessage`	string	The error message that appears if the lookup filter fails.
`informationalMessage`	string	The information message displayed on the page. Use to describe things the user might not understand, such as why certain items are excluded in the lookup filter.
`name`	string	Required. The name of the lookup filter. If you create this field in the user interface, a name is automatically assigned.

Field	Field Type	Description
		If you create this field through the Metadata API, you must include the `name` field.

ObjectNameCaseValue

ObjectNameCaseValue supports multiple cases and definitions of the custom object name to allow usage in various grammatical contexts.

Field	Field Type	Description
`article`	Article (enumeration of type string)	English has two types of articles: definite (*`the`*) and indefinite (*`a`*, *`an`*). The usage of these articles depends mainly on whether you are referring to any member of a group, or to a specific member of a group. The valid values are: • `Definite` • `Indefinite` • `None`
`caseType`	CaseType (enumeration of type string)	The case of the custom object name. The valid values are: • `Ablative` • `Accusative` • `Adessive` • `Allative` • `Causalfinal` • `Dative` • `Delative` • `Distributive` • `Elative` • `Essive` • `Essiveformal` • `Genitive` • `Illative` • `Inessive` • `Instrumental` • `Locative` • `Nominative` • `Objective` • `Partitive` • `Prepositional` • `Subjective` • `Sublative` • `Superessive` • `Termanative` • `Translative` • `Vocative`

Field	Field Type	Description
plural	boolean	Indicates whether the value field is plural (true) or singular (false).
possessive	Possessive (enumeration of type string)	The possessive case of a language is a grammatical case used to indicate a relationship of possession. The valid values are: • First • None • Second
value	string	Required. The value or label in this grammatical context.

PicklistValueTranslation

PicklistValueTranslation contains details for a picklist value translation. For more details, see Picklist on page 108.

Field	Field Type	Description
masterLabel	string	Required. The picklist value defined on the setup page in the application is your master label. The master label is displayed wherever a translated label is not available.
translation	string	Required. Translation for the value.

RecordTypeTranslation

RecordTypeTranslation contains details for a record type name translation. For more details, see RecordType on page 114.

Field	Field Type	Description
label	string	Required. Translation for the label. Maximum of 765 characters.
name	string	Required. The record type name.

SharingReasonTranslation

SharingReasonTranslation contains details for a sharing reason translation. For more details, see SharingReason on page 118.

Field	Field Type	Description
label	string	Required. Translation for the sharing reason.
name	string	Required. The sharing reason name.

ValidationRuleTranslation

ValidationRuleTranslation contains details for a validation rule translation. For more details, see ValidationRule on page 119.

Field	Field Type	Description
errorMessage	string	Required. Translation for the error message associated with the validation rule failure.

Field	Field Type	Description
`name`	string	Required. The validation rule name.

WebLinkTranslation

WebLinkTranslation contains details for a web link translation. For more details, see Weblink on page 120.

Field	Field Type	Description
`label`	string	Required. Translation for the web link label. Maximum of 765 characters.
`name`	string	Required. The web link name.

WorkflowTaskTranslation

WorkflowTaskTranslation contains details for a workflow task translation. For more details, see Workflow on page 210.

Field	Field Type	Description
`description`	string	Translation for the workflow task description.
`name`	string	Required. The workflow task name.
`subject`	string	Translation for the workflow task subject.

Declarative Metadata Sample Definition

A sample XML definition of a custom object translation is shown below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<CustomObjectTranslation xmlns="http://soap.sforce.com/2006/04/metadata">
    <fields>
        <fullName>Description__c</fullName>
        <label>description</label>
    </fields>
    <gender>Feminine</gender>
    <nameFieldLabel>citation</nameFieldLabel>
    <objectNames>
        <article>None</article>
        <value>description</value>
    </objectNames>
    <objectNames>
        <article>Indefinite</article>
        <value>une description</value>
    </objectNames>
    <objectNames>
        <article>Definite</article>
        <value>la description</value>
    </objectNames>
    <objectNames>
        <article>None</article>
        <plural>true</plural>
        <value>descriptions </value>
    </objectNames>
    <objectNames>
        <article>Definite</article>
        <plural>true</plural>
        <value>les descriptions</value>
    </objectNames>
    <startsWith>Consonant</startsWith>
</CustomObjectTranslation>
```

See Also:

CustomObject
Translations

CustomPageWebLink

Represents a web link defined in a home page component. It extends the Metadata metadata type and inherits its `fullName` field. All other web links are stored as a Weblink in a CustomObject.

Declarative Metadata File Suffix and Directory Location

There is one file per web link definition, stored in the `weblinks` folder in the corresponding package directory. The file suffix is `.weblink`.

Version

CustomPageWebLinks are available in API version 13.0 and later.

Fields

The CustomPageWebLink definition has the following fields:

Field Name	Field Type	Description
`availability`	WebLinkAvailability (enumeration of type string)	Required. Indicates whether the Weblink is only available online (`online`, or if it is also available offline (`offline`).
`description`	string	A description of the Weblink.
`displayType`	WebLinkDisplayType (enumeration of type string)	Represents how this Weblink is rendered. Valid values: • `link` for a hyperlink • `button` for a button • `massAction` for a button attached to a related list
`fullName`	string	The name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
`hasMenubar`	boolean	If the `openType` is `newWindow`, whether to show the browser menu bar for the window (`true` or not (`false`). Otherwise this field should not be specified.
`hasScrollbars`	boolean	If the `openType` is `newWindow`, whether to show the scroll bars for the window (`true`) or not (`false`). Otherwise this field should not be specified.

Field Name	Field Type	Description
hasToolbar	boolean	If the openType is newWindow, whether to show the browser toolbar for the window (true) or not (false). Otherwise this field should not be specified.
height	int	Height in pixels of the window opened by this Weblink. Required if the openType is newWindow, otherwise should not be specified
isResizable	boolean	If the openType is newWindow, whether to allow resizing of the window (true) or not (false). Otherwise this field should not be specified.
linkType	WebLinkType (enumeration of type string)	Required. Represents whether the content of this Weblink is specified by a URL, an sControl, or by a JavaScript code block. Valid values: • url • sControl • javascript • page
masterLabel	string	The master label for the Weblink.
openType	WebLinkWindowType (enumeration of type string)	Required. When this button is clicked, specifies the window style that will be used to display the content. Valid values: • newWindow • sidebar • noSidebar • replace • onClickJavaScript
page	string	If the value of linkType is page, this field represents the Visualforce page; otherwise, this field should not be specified.
position	WebLinkPosition (enumeration of type string)	If the openType is newWindow, how the new window should be displayed. Otherwise this field should not be specified. Valid values: • fullScreen • none • topLeft
protected	boolean	Required. Indicates whether this component is protected (true) or not (false). Protected components cannot be linked to or referenced by components created in the installing organization.
requireRowSelection	boolean	If the openType is massAction, whether to require individual row selection to execute the action for this button (true) or not (false). Otherwise this field should not be specified.
scontrol	string	If the value of linkType is sControl, this field represents the name of the sControl; otherwise, this field should not be specified.

Field Name	Field Type	Description
showsLocation	boolean	If the openType is newWindow, whether or not to show the browser location bar for the window; otherwise this field should not be specified.
showsStatus	boolean	If the openType is newWindow, whether or not to show the browser status bar for the window. Otherwise, this field should not be specified.
url	string	If the value of linkType is url, this is the URL value. If the value of linkType is javascript, this is the JavaScript content. If the value neither of these, the this field should not be specified. Content must be escaped in a manner consistent with XML parsing rules.
width	int	Width in pixels of the window opened by this Weblink. Required if the openType is newWindow, otherwise should not be specified.

Declarative Metadata Sample Definition

The following is the definition of a Weblink. For related samples, see HomePageComponent on page 161 and HomePageLayout on page 162.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomPageWebLink xmlns="http://soap.sforce.com/2006/04/metadata">
    <availability>online</availability>
    <displayType>button</displayType>
    <hasMenubar>false</hasMenubar>
    <hasScrollbars>true</hasScrollbars>
    <hasToolbar>false</hasToolbar>
    <height>600</height>
    <isResizable>true</isResizable>
    <linkType>url</linkType>
    <masterLabel>detailPageButon</masterLabel>
    <openType>newWindow</openType>
    <position>none</position>
    <protected>false</protected>
    <showsLocation>false</showsLocation>
    <showsStatus>false</showsStatus>
    <url>http://google.com</url>
</CustomPageWebLink>
```

See Also:

HomePageComponent
HomePageLayout
Weblink

CustomSite

Force.com sites enables you to create public websites and applications that are directly integrated with your Salesforce.com organization—without requiring users to log in with a username and password. For more information, see “Sites Overview” in the Salesforce.com online help.

Note: CustomSite does not support syndication feeds at this time.

Declarative Metadata File Suffix and Directory Location

Force.com CustomSite components are stored in the `sites` directory of the corresponding package directory. The file name matches the site name, and the extension is `.site`.

Version

Force.com CustomSite components are available in API version 14.0 and later.

Fields

Field	Field Type	Description
`active`	boolean	Required. Determines whether or not the site is active.
`allowHomePage`	boolean	Determines whether or not the standard home page is visible to public users. This is a new field in API version 15.0.
`allowStandardIdeasPages`	boolean	Determines whether or not the standard Salesforce CRM Ideas pages are visible to public users. This is a new field in API version 15.0.
`allowStandardLookups`	boolean	Determines whether or not the standard lookup pages are visible to public users. This is a new field in API version 15.0.
`allowStandardSearch`	boolean	Determines whether or not the standard search pages are visible to public users. This is a new field in API version 15.0.
`analyticsTrackingCode`	string	The tracking code associated with your site. This code can be used by services like Google Analytics to track page request data for your site. This field is available in API version 17.0 and later.
`authorizationRequiredPage`	string	The name of the Visualforce page to be displayed when the guest user tries to access a page for which they are not authorized.
`bandwidthExceededPage`	string	The name of the Visualforce page to be displayed when the site has exceeded its bandwidth quota.
`changePasswordPage`	string	The name of the Visualforce page to be displayed when the portal user attempts to change his or her password.
`customWebAddress`	string	The custom Web address associated with the site.

Field	Field Type	Description
`description`	string	The site description.
`favoriteIcon`	string	The name of the file to be used for the icon that appears in the browser's address field when visiting the site. Sets the favorite icon for the entire site.
`fileNotFoundPage`	string	The name of the Visualforce page to be displayed when the guest user tries to access a non-existent page.
`genericErrorPage`	string	The name of the Visualforce page to be displayed for errors not otherwise specified.
`guestProfile`	string	Read only. The name of the profile associated with the guest user.
`inMaintenancePage`	string	The name of the Visualforce page to be displayed when the site is down for maintenance.
`inactiveIndexPage`	string	The name of the Visualforce page set as the inactive site home page.
`indexPage`	string	Required. The name of the Visualforce page set as the active site home page.
`masterLabel`	string	The name of the site label in the Salesforce.com user interface.
`portal`	string	The name of the portal associated with this site for login access.
`requireInsecurePortalAccess`	string	Required. Determines whether to override your organization's security settings and exclusively use HTTP when logging in to the associated portal from your site.
`robotsTxtPage`	string	The name of the Visualforce page to be displayed for the `robots.txt` file used by web crawlers.
`serverIsDown`	string	The name of the static resource to be displayed from the cache server when Salesforce.com servers are down. The static resource must be a public zip file 1 MB or smaller and must contain a page named `maintenance.html` at the root level of the zip file. Other resources in the zip file, such as images or CSS files, can follow any directory structure. This field is available in API version 17.0 and later.
`siteRedirectMappings`	SiteRedirectMapping[]	An array of all URL redirect rules set for your site. This field is available in API version 20.0 and later.
`siteAdmin`	string	The username of the site administrator.
`siteTemplate`	string	The name of the Visualforce page to be used as the site template.
`subdomain`	string	Required. Read only. The custom subdomain prefix for the site. For example, if your site URL is `mycompany.force.com/partners`, `mycompany.force.com` is the subdomain.

Field	Field Type	Description
`urlPathPrefix`	string	The first part of the path on the site's URL that distinguishes this site from other sites. For example, if your site URL is `mycompany.force.com/partners`, `partners` is the urlPathPrefix.

SiteRedirectMapping

SiteRedirectMapping represents a URL redirect rule on your Force.com site. For more information, see "Sites URL Redirects" in the Salesforce.com online help.

Field	Field Type	Description
`action`	SiteRedirect (enumeration of type string)	The type of the redirect. Available string values are: • Permanent • Temporary
`isActive`	boolean	The status of the redirect: active or inactive.
`source`	string	The URL that you want to redirect. It must be a relative URL, but can have any valid extension type, such as `.html` or `.php`.
`target`	string	The new URL you want users to visit. It can be a relative URL or a fully-qualified URL with an `http://` or `https://` prefix.

Declarative Metadata Sample Definition

A sample XML definition of a site is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomSite xmlns="http://soap.sforce.com/2006/04/metadata">
    <active>true</active>
    <allowHomePage>true</allowHomePage>
    <allowStandardIdeasPages>true</allowStandardIdeasPages>
    <allowStandardLookups>true</allowStandardLookups>
    <allowStandardSearch>true</allowStandardSearch>
    <authorizationRequiredPage>Unauthorized</authorizationRequiredPage>
    <bandwidthExceededPage>BandwidthExceeded</bandwidthExceededPage>
    <changePasswordPage>ChangePassword</changePasswordPage>
    <favoriteIcon>myFavIcon</favoriteIcon>
    <fileNotFoundPage>FileNotFound</fileNotFoundPage>
    <genericErrorPage>Exception</genericErrorPage>
    <inMaintenancePage>InMaintenance</inMaintenancePage>
    <serverIsDown>MyServerDownResource</serverIsDown>
    <indexPage>UnderConstruction</indexPage>
    <masterLabel>customSite</masterLabel>
    <portal>Customer Portal</portal>
    <requireInsecurePortalAccess>false</requireInsecurePortalAccess>
    <siteAdmin>admin@myco.org</siteAdmin>
    <siteTemplate>SiteTemplate</siteTemplate>
    <subdomain>myco</subdomain>
</CustomSite>
```

See Also:

Portal

CustomTab

A custom tab is a user interface component you create to display custom object data or other web content embedded in the application. When a tab displays a custom object, the tab name is the same as the custom object name; for page, s-control, or URL tabs, the name is arbitrary. For more information, see "What is a Custom Tab?" in the Salesforce.com online help. It extends the Metadata metadata type and inherits its `fullName` field.

File Suffix and Directory Location

The file suffix is `.tab`. There is one file for each tab, stored in the `tabs` folder in the corresponding package directory.

Note: Retrieving a component of this metadata type in a project makes the component appear in the Profile component as well.

Version

Tabs are available in API version 10.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`customObject`	boolean	Indicates whether this tab is for a custom object (`true`) or not (`false`). If set to `true`, the name of the tab matches the name of the custom object. Only one of these fields should have a value set: • `customObject` • `page` • `scontrol` • `url`
`description`	string	The optional description text for the tab.
`frameHeight`	int	The height, in pixels of the tab frame. Required for s-control and page tabs.
`fullName`	string	The name of the tab. The value of this field depends on the type of tab, and the API version. • For custom object tabs, the `fullName` is the developer-assigned name of the custom object (MyCustomObject__c, for example).

Field Name	Field Type	Description
		For custom object tabs, this name must be the same as the custom object name, and `customObject` should be set to `true`. • For Web tabs, the `fullName` is the developer-assigned name of the tab (MyWebTab, for example). The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`hasSidebar`	boolean	Indicates if the tab displays the sidebar panel.
`icon`	string	The optional reference to the image document for the tab if the tab is not using one of the standard tab styles. This is a new field in API version 14.0.
`label`	string	This is the label of the tab, for Web tabs only.
`mobileReady`	boolean	Required. Indicates if the custom tab is available for Mobile Edition (`true`) or not (`false`).
`motif`	string	Required. The tab style for the color scheme and icon for the custom tab. For example, "'Custom70: Handsaw," is the handsaw icon.
`page`	string	The name of the Visualforce page to display in this tab. Only one of these fields should have a value set: • `customObject` • `page` • `scontrol` • `url`
`scontrol`	string	The name of the s-control to display in this tab. Only one of these fields should have a value set: • `customObject` • `page` • `scontrol` • `url`
`splashPageLink`	string	The custom link used as the introductory splash page when users click the tab. References a HomePageComponent.
`url`	string	The URL for the external web-page to embed in this tab. Only one of these fields should have a value set: • `customObject` • `page` • `scontrol` • `url`
`urlEncodingKey`	Encoding (enumeration of type string)	The default encoding setting is Unicode: `UTF-8`. Change it if you are passing information to a URL that requires data in a different

Field Name	Field Type	Description
		format. This option is available when the value `URL` is selected in the tab type.

Declarative Metadata Sample Definition

The following is the definition of a tab:

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomTab xmlns="http://soap.sforce.com/2006/04/metadata">
    <description>Myriad Publishing</description>
    <frameHeight>600</frameHeight>
    <mobileReady>true</mobileReady>
    <motif>Custom53: Bell</motif>
    <url>http://www.myriadpubs.com</url>
    <urlEncodingKey>UTF-8</urlEncodingKey>
</CustomTab>
```

See Also:

CustomApplication

Dashboard

Dashboards are visual representations of data that allow you to see key metrics and performance at a glance. It extends the Metadata metadata type and inherits its `fullName` field. For more information, see "Editing Dashboards" in the Salesforce.com online help.

Declarative Metadata File Suffix and Directory Location

Dashboards are stored in the `dashboards` directory of the corresponding package directory. The file name matches the dashboard title and the extension is `.dashboard`.

Version

Dashboard components are available in API version 14.0 and later.

Fields

Field	Field Type	Description
`backgroundEndColor`	string	Required. A dashboard can have a gradient color change on its charts. This field defines the second color for the gradient and `backgroundStartColor` defines the first color. If you prefer your background to be all one color or do not want a gradient color change, select the same color for this field and `backgroundStartColor`. The color is in hexadecimal format; for example #FF6600.

Field	Field Type	Description
`backgroundFadeDirection`	ChartBackgroundDirection (enumeration of type string)	Required. The direction of the gradient color change, defined by the `backgroundStartColor` and `backgroundEndColor` fields. The valid values are: • `diagonal` • `leftToRight` • `topToBottom`
`backgroundStartColor`	string	Required. The starting color for the gradient color change on the dashboard's charts. See `backgroundEndColor` for more information. The color is in hexadecimal format; for example #FF6600.
`dashboardType`	DashboardType (enumeration of type string)	Determines the way visibility settings are set for a dashboard. The valid values are: • `SpecifiedUser`—All users see data at the access level of one specific running user, specified in the `runningUser` field, regardless of their own security settings. • `LoggedInUser`—Each logged-in user sees data according to his or her own access level. • `MyTeamUser`—Managers can choose to view the dashboard from the point of view of their subordinates in the role hierarchy. This value is available in API version 20.0 and later. This field is available in API version 19.0 and later.
`description`	string	Description for the dashboard. Maximum of 255 characters.
`fullName`	string	Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call. This field specifies the folder and dashboard title; for example `folderSales/California`.
`leftSection`	Section	Required. The left section or column of the dashboard.
`middleSection`	Section	The middle section or column of the dashboard.
`rightSection`	Section	Required. The right section or column of the dashboard.
`runningUser`	string	The username of the user whose role and sharing settings are used to determine the data shown in the dashboard. When you deploy a dashboard and the value in this field is not defined or does not correspond to a valid user, the field is populated with the username of the user performing the deployment. Regardless of their security settings, all users viewing a dashboard see exactly the same data, because dashboards are always run using the security settings of a particular user.

Field	Field Type	Description
		Tip: To avoid inappropriate exposure of sensitive data, save the dashboard to a folder that is visible only to appropriate users.
`textColor`	string	Required. Color of the text on each chart in the dashboard. The color is in hexadecimal format; for example #FF6600.
`title`	string	Required. The dashboard title.
`titleColor`	string	Required. Color of the titles on each dashboard component. The color is in hexadecimal format; for example #FF6600.
`titleSize`	int	Required. Size of characters in title text. For example, a value of 12 indicates 12pt text.

Section

DashboardComponentSection represents one of the sections or columns in a dashboard.

Field	Field Type	Description
`columnSize`	DashboardComponentSize (enumeration of type string)	Required. The size of the column in the dashboard. See DashboardComponentSize for details on valid values.
`components`	DashboardComponent[]	The list of DashboardComponent objects in the dashboard column.

DashboardComponentSize

DashboardComponentSize is an enumeration of type string that lists different size categories. The valid values are listed in the table below:

Enumeration Value	Description
`medium`	Medium component size.
`narrow`	Smallest component size.
`wide`	Largest component size.

DashboardComponent

A dashboard consists of a group of different components or elements that display data. Each component can use a custom report or a custom s-control as their data source to display corporate metrics or key performance indicators. You can create several dashboard components and display them all in one dashboard aligned in up to three columns.

Field	Field Type	Description
`chartAxisRange`	ChartRangeType (enumeration of type string)	A manual or automatic axis range for bar or line charts. The valid values are: • `auto` • `manual`

Field	Field Type	Description
chartAxisRangeMax	double	The maximum axis range to be displayed. This only applies to bar and line charts in which the manual axis range is selected for the chartAxisRange field.
chartAxisRangeMin	double	The minimum axis range to be displayed. This only applies to bar and line charts in which the manual axis range is selected for the chartAxisRange field.
componentType	DashboardComponentType (enumeration of type string)	Required. Dashboard component type. The valid values are: • Bar • BarGrouped • BarStacked • BarStacked100 • Column • ColumnGrouped • ColumnStacked • ColumnStacked100 • Donut • Funnel • Gauge • Line • lineCumulative • LineGrouped • lineGroupedCumulative • Metric • Pie • Scontrol • Table
dashboardTableColumn	DashboardTableColumn[]	Represents a list of columns on a customized dashboard table component.
displayUnits	ChartUnits (enumeration of type string)	Chart Units. The valid values are: • Auto • Integer • Hundreds • Thousands • Millions • Billions • Trillions
drillDownUrl	string	For charts, specifies a URL that users go to when they click the dashboard component. Use this option to send users to another dashboard, report, record detail page, or other system that uses a Web interface. This field overrides the drillEnabled and drillToDetailEnabled fields.

Field	Field Type	Description
`drillEnabled`	boolean	Specifies whether to take users to the full or filtered source report when they click the dashboard component. Set to `false` to drill to the full source report; set to `true` to drill to the source report filtered by what they clicked. If set to `true`, users can click individual groups, axis values, or legend entries. This overrides the `drillToDetailEnabled` field. This field is available in API version 17.0 and later.
`drillToDetailEnabled`	boolean	When enabled, users are taken to the record detail page when they click a record name, record owner, or feed post in a table or chart. When set to `true` users can click axis and legend values, chart elements, and table entries. The `drillDownUrl` and `drillEnabled` fields override this field. This field is available in API version 20.0 and later.
`enableHover`	boolean	Specifies whether to display values, labels, and percentages when hovering over charts. Hover details depend on chart type. Percentages apply to pie, donut, and funnel charts only. This field is available in API version 17.0 and later.
`expandOthers`	boolean	Specifies whether to combine all groups less than or equal to 3% of the total into a single 'Others' wedge or segment. This only applies to pie, donut, and funnel charts. Set to `true` to show all values individually on the chart; set to `false` to combine small groups into 'Others.' This field is available in API version 17.0 and later.
`footer`	string	Footer displayed at the bottom of the dashboard component. Maximum of 255 characters.
`gaugeMax`	double	The maximum value on a gauge. A gauge is used to see how far you are from reaching a goal. It looks like a speedometer in a car.
`gaugeMin`	double	The minimum value on a gauge.
`header`	string	Header displayed at the top of the dashboard component. Maximum of 80 characters.
`indicatorBreakpoint1`	double	The value that separates the `indicatorLowColor` from the `indicatorMiddleColor` on the dashboard.
`indicatorBreakpoint2`	double	The value that separates the `indicatorMiddleColor` from the `indicatorHighColor` on the dashboard.
`indicatorHighColor`	string	The color representing a high number range on the gauge.
`indicatorLowColor`	string	The color representing a low number range on the gauge.
`indicatorMiddleColor`	string	The color representing a medium number range on the gauge.

Field	Field Type	Description
`legendPosition`	ChartLegendPosition (enumeration of type string)	The location of the legend with respect to the chart. The valid values are: • `Bottom` • `OnChart` • `Right`
`maxValuesDisplayed`	int	The maximum number of elements to include in the top-level grouping of the horizontal axis of a horizontal chart, vertical axis of a vertical chart, or selected axis of a stacked bar chart. For example, if you want to list only your top five salespeople, create an opportunity report that lists total opportunity amounts by owner and enter `5` in this field.
`metricLabel`	string	Descriptive label for the metric. This is relevant if `metric` is the value of the `componentType` field.
`page`	string	Visualforce page associated with the component.
`pageHeightInPixels`	int	Display height of the Visualforce page in pixels.
`report`	string	Name of the report associated with the component.
`scontrol`	string	S-control associated with component if `scontrol` is the value of the `componentType` field. For more information, see "Defining Custom S-Controls" in the Salesforce.com online help.
`scontrolHeightInPixels`	int	Display height of the s-control in pixels.
`showPercentage`	boolean	Indicates if percentages are displayed for regions of gauges and wedges and segments of pie, donut, and funnel charts (`true`), or not (`false`).
`showTotal`	boolean	Indicates if the total of all wedges is displayed for gauges and donut charts (`true`), or not (`false`).
`showValues`	boolean	Indicates if the values of individual records or groups are displayed for charts (`true`), or not (`false`).
`sortBy`	DashboardComponentFilter (enumeration of type string)	The sort option for the dashboard component.
`title`	string	The title of the dashboard component. Maximum of 40 characters.
`useReportChart`	boolean	Specifies whether to use the chart defined in the source report on this dashboard component. The chart settings in the source report determine how the chart displays in the dashboard, and any chart settings you define for the dashboard are overridden. If you defined a combination chart in the source report, use this option to use that combination chart on this dashboard.

DashboardTableColumn

DashboardTableColumn represents a column in a customized table component in a dashboard.

Field	Field Type	Description
aggregateType	ReportSummaryType[] (enumeration of type string)	Specifies the aggregation type for the table column.
column	string	Required. The label of the column to use in the table.
showTotal	boolean	Displays the totals for each summarizable column in the dashboard table. This field is available in API version 19.0 and later.
sortBy	DashboardComponentFilter (enumeration of type string)	The sort option for the dashboard table component. Sort on just one column per table.

DashboardComponentFilter

DashboardComponentFilter is an enumeration of type string that lists the sort values for dashboard components. The valid values are:

Enumeration Value	Description
RowLabelAscending	Sorts in alphabetical order by the label.
RowLabelDescending	Sorts in reverse alphabetical order by the label.
RowValueAscending	Sorts lowest to highest by the value.
RowValueDescending	Sorts highest to lowest by the value.

Declarative Metadata Sample Definition

A sample XML definition of a dashboard is shown below. The file name matches the dashboard title and the extension is `.dashboard`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Dashboard xmlns="http://soap.sforce.com/2006/04/metadata">
    <backgroundEndColor>#FFFFFF</backgroundEndColor>
    <backgroundFadeDirection>LeftToRight</backgroundFadeDirection>
    <backgroundStartColor>#FFFFFF</backgroundStartColor>
    <description>Dashboard with all possible chart types</description>
    <leftSection>
        <columnSize>Medium</columnSize>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>BarStacked100</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <componentType>Table</componentType>
            <dashboardTableColumn>
                <column>CLOSE_DATE</column>
                <sortBy>RowLabelAscending</sortBy>
            </dashboardTableColumn>
            <dashboardTableColumn>
                <aggregateType>Sum</aggregateType>
                <column>AMOUNT</column>
                <showTotal>true</showTotal>
            </dashboardTableColumn>
            <dashboardTableColumn>
                <column>STAGE_NAME</column>
            </dashboardTableColumn>
            <dashboardTableColumn>
                <column>PROBABILITY</column>
                <aggregateType>Maximum</aggregateType>
            </dashboardTableColumn>
            <displayUnits>Integer</displayUnits>
            <header>Opportunities Table</header>
            <indicatorHighColor>#54C254</indicatorHighColor>
            <indicatorLowColor>#C25454</indicatorLowColor>
            <indicatorMiddleColor>#C2C254</indicatorMiddleColor>
            <maxValuesDisplayed>10</maxValuesDisplayed>
            <report>testFolder/sourceRep</report>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>Bar</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>Column</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <legendPosition>Bottom</legendPosition>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
            <useReportChart>true</useReportChart>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>Funnel</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <expandOthers>true</expandOthers>
            <legendPosition>Bottom</legendPosition>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
    </leftSection>
    <middleSection>
        <columnSize>Medium</columnSize>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>ColumnStacked100</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>ColumnStacked</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>ColumnStacked</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>ColumnGrouped</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>Column</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
    </middleSection>
    <rightSection>
        <columnSize>Medium</columnSize>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>Bar</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>Pie</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <expandOthers>true</expandOthers>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>LineGroupedCumulative</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>LineGrouped</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>LineCumulative</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
        <components>
            <chartAxisRange>Auto</chartAxisRange>
            <componentType>Donut</componentType>
            <displayUnits>Auto</displayUnits>
            <drillEnabled>true</drillEnabled>
            <enableHover>true</enableHover>
            <expandOthers>true</expandOthers>
            <report>testFolder/sourceRep</report>
            <sortBy>RowLabelAscending</sortBy>
        </components>
    </rightSection>
    <runningUser>admin@TESTORGNUM</runningUser>
    <textColor>#000000</textColor>
    <title>Db Title</title>
    <titleColor>#000000</titleColor>
    <titleSize>12</titleSize>
</Dashboard>
```

See Also:

Folder
Report

DataCategoryGroup

Represents a data category group. It extends the Metadata metadata type and inherits its `fullName` field.

Caution: Using Metadata API to deploy category changes from one organization to another permanently removes categories and record categorizations that are not specified in your XML file. Salesforce.com recommends that you manually create data categories and record associations in an organization using ***`Your Name`*** ➤ **Setup** ➤ **Customize** ➤ **Data Categories** rather than deploying changes from a sandbox to a production organization. For more information see Usage on page 151.

Data category groups are provided to:

- Classify and filter data.
- Share data among users.

Every data category group contains items or data categories that can be organized hierarchically.

The example below shows the `Geography` data category group and its data categories.

```
Geography
    Worldwide
        North America
            United States of America
            Canada
            Mexico
        Europe
        Asia
```

Note: See "What are Data Categories?" in the Salesforce.com online help for more information on data category groups, data categories, parent and sub categories.

File Suffix and Directory Location

The file suffix is `.datacategorygroup`. There is one file for each data category group stored in the `datacategorygroups` folder in the corresponding package directory.

Version

Data category groups are available in API version 18.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`active`	boolean	Required. The status of the category group. Indicates whether this category group is active, (`true`), or not active (`false`).
`dataCategory`	DataCategory	Required. The top-level category within the data category group.
`description`	string	The description of the data category group.
`fullName`	string	Required. The unique name of the data category group. When creating a data category group, the `fullName` field and the file name (without its suffix) must match.The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`label`	string	Required. Label that represents the object in Salesforce.com.
`objectUsage`	ObjectUsage	The objects that are associated with the data category group.

DataCategory

Represents an item (or data category) in the data category group. A data category can recursively contain a list of other data categories.

Field Name	Field Type	Description
`dataCategory`	DataCategory[]	A recursive list of sub data categories. For example, a list of countries within a continent. By default, for each category group you can create up to 100 categories and organize those categories into up to fivehierarchy levels
`label`	string	Required. Label for the data category throughout the Salesforce.com user interface.

Field Name	Field Type	Description
name	string	Required. The developer name of the data category used as a unique identifier for API access. The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. **Important:** The value for this field is defined once and cannot be changed later. **Caution:** If you deploy a category group that already exists in an organization, any category that is not defined in the XML file is permanently removed from your organization. For more information see Usage on page 151.

ObjectUsage

Represents the objects that can be associated with the data category group. This association allows the object to be classified and filtered using the data categories.

Field Name	Field Type	Description
object	string[]	A list of the object names that can be associated with the data category group. Valid values are: • `KnowledgeArticleVersion`—to associate articles. See "Modifying Category Group Assignments in Salesforce Knowledge" in the Salesforce.com online help for more information on data category groups association to articles. • `Question`—to associate questions. You can associate the `Question` object with at most one category group. See "Assigning Data Categories to Answers" in the Salesforce.com online help for more information on data category groups association to questions. **Caution:** If you deploy a category group that already exists in an organization, any object association that is not defined in the XML file is permanently removed from your organization. Ensure that your XML file specifies all the records associated with your category group in the organization. For more information see Usage on page 151.

Declarative Metadata Sample Definition

This sample is the definition of the `Geography` data category group and its data categories:

```
<?xml version="1.0" encoding="UTF-8"?>
<DataCategoryGroup xmlns="http://soap.sforce.com/2006/04/metadata">
   <label>Geography</label>
   <description>Geography structure of service center locations</description>
   <fullName>geo</fullName>
   <dataCategory> <name>WW</name> <label>Worldwide</label>
      <dataCategory> <name>AMER</name> <label>North America</label>
         <dataCategory>
             <name>USA</name>
             <label>United States of America</label>
         </dataCategory>
         <dataCategory>
             <name>CAN</name>
             <label>Canada</label>
         </dataCategory>
         <dataCategory>
             <name>MEX</name>
             <label>Mexico</label>
         </dataCategory>
      </dataCategory>
      <dataCategory> <name>EMEA</name> <label>Europe, Middle East, Africa</label>
         <dataCategory>
             <name>FR</name>
             <label>France</label>
         </dataCategory>
         <dataCategory>
             <name>SP</name>
             <label>Spain</label>
        </dataCategory>
         <dataCategory>
            <name>UK</name>
            <label>United-Kingdom</label>
         </dataCategory>
      </dataCategory>
      <dataCategory>
          <name>APAC</name>
          <label>Asia</label>
      </dataCategory>
   </dataCategory>

   <objectUsage>
      <object>KnowledgeArticleVersion </object>
   <objectUsage>
</DataCategoryGroup>
```

Usage

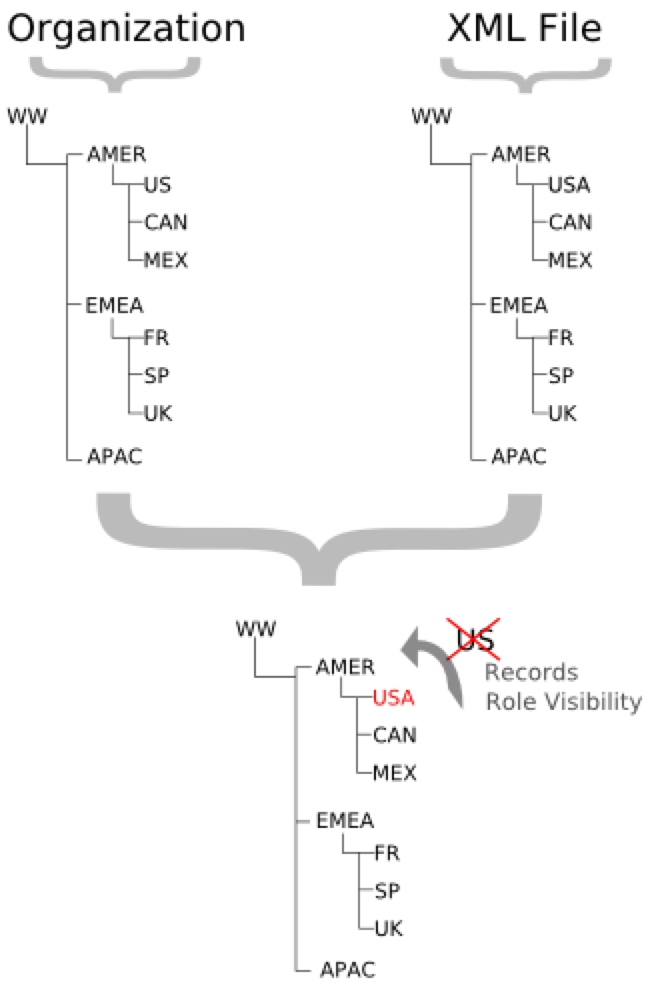
When you deploy a category group XML file, the Metadata API checks whether the category group exists in the target organization. If the category group does not exist, it is created. If the category group already exists, then the Metadata API:

- Adds any new category or object defined in the XML file.
- Deletes any category that is not defined in the XML file. Records associated with the deleted categories are re-associated with the parent category.
- Deletes any object association that is not defined in the XML file.
- Moves any category if its hierarchical position differs from the position specified in the XML file. When a category moves to a new parent category, roles that have no visibility on the new parent category lose their visibility to the repositioned category.

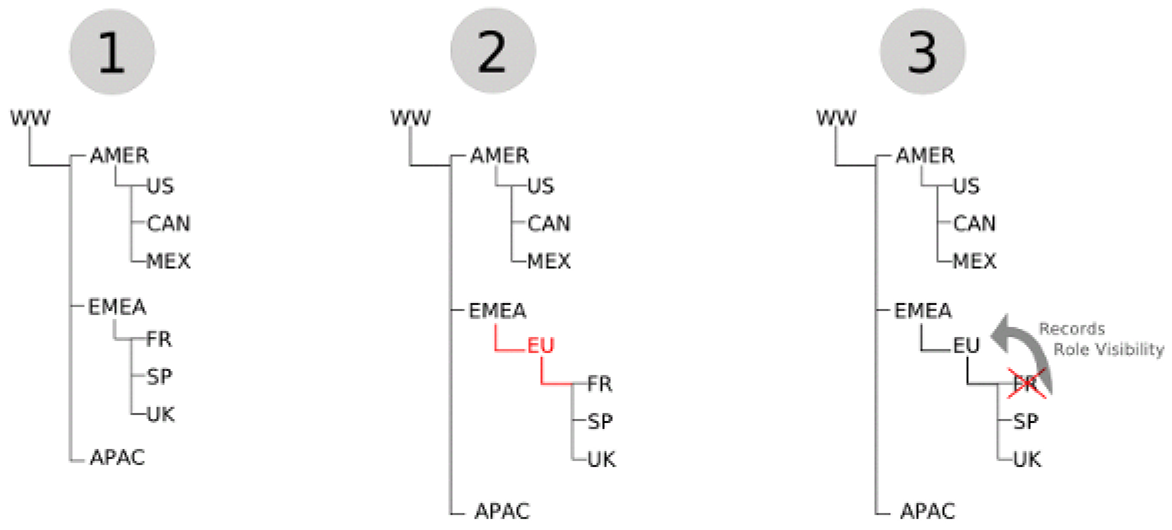
Note: For more information about category deletion, category repositioning and its impact on record categorization and role visibility see "Deleting Data Categories" and "Modifying and Positioning Data Categories" in the Salesforce.com online help.

Using Metadata API to deploy category changes from one organization to another permanently removes categories and record categorizations that are not specified in your XML file. Salesforce.com recommends that you manually create data categories and record associations in an organization using ***`Your Name`*** ➤ **Setup** ➤ **Customize** ➤ **Data Categories** rather than deploying changes from a sandbox to a production organization.

The following example illustrates what happens if you deploy an XML representation of a `Geography` data category group hierarchy to an organization that already has this data category group defined. Note that the organization contains a `US` category, while the XML file includes a `USA` category in the same hierarchical position. The Metadata API deployment process deletes the `US` category from the organization and moves associations for any records from `US` to the parent `AMER` category. It also adds the `USA` category under `AMER`. Note that all records that were previously categorized with `US` are now associated with the `AMER` category.



The next example illustrates what can happen when you delete or move a category in a data category group and deploy its XML representation from a sandbox to a production organization that already has this data category group defined. Hierarchy 1 shows the initial data category group in the sandbox organization. In hierarchy 2, we add an `EU` category under `EMEA` and move `FR`, `SP` and `UK` below `EU`. In hierarchy 3, we delete `FR` and associate its records with its new parent, `EU`. Finally, we deploy the changes from the sandbox to the production organization.



The Metadata API has no concept of the order of the changes made to the sandbox organization. It just deploys the changes from one organization to another. During the deployment, it first notices the deletion of the `FR` category and removes it from the production organization. Consequently, it moves associations for any records from `FR` to its parent on the production organization, `EMEA`. The Metadata API then adds the `EU` category and moves `SP` and `UK` below it. Although the category group hierarchy looks the same in both organizations, record categorization in production is different from the sandbox organization. The records that were originally associated with `FR` in hierarchy 1 are associated with `EU` in the sandbox organization, but are associated with `EMEA` in the production organization.

Document

Represents a Document. All documents must be in a document folder, for example `sampleFolder/TestDocument`. This metadata type extends the MetadataWithContent component and shares its fields.

Currently, users are not able to export document metadata to a local file system using the Force.com IDE.

Version

Documents are available in API version 10.0 and later.

In API version 17.0 and later, you can delete a folder containing documents moved to the Recycle Bin. When you delete the folder, any related documents in the Recycle Bin are permanently deleted.

In API version 18.0 and later, documents do not need an extension.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`content`	base64	Content of the document. Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion

Field Name	Field Type	Description
		is usually handled for you by a SOAP client. This field is inherited from the MetadataWithContent component.
`description`	string	A description of the document. Enter a description to distinguish this document from others.
`fullName`	string	The name of the document, including the folder name. In version 17.0 and earlier, the `fullName` included the document extension. In version 18.0 and later, the `fullName` does not include the file extension. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. If this field contained characters before version 14.0 that are no longer allowed, the characters were stripped out of this field, and the previous value of the field was saved in the `name` field. This field is inherited from the Metadata component.
`internalUseOnly`	boolean	Required. Indicates whether the document is confidential (`true`) or not (`false`). This field and `public` are mutually exclusive; you cannot set both to `true`.
`keywords`	string	Contains one or more words that describe the document. A check for matches to words in this field is performed when doing a search.
`name`	string	The list of characters allowed in the `fullName` field has been reduced for versions 14.0 and later. This field contains the value contained in the `fullName` field before version 14.0. This field is only populated if the value of the `fullName` field contained characters that are no longer accepted in that field.
`public`	boolean	Required. Indicates whether the document is an image available for HTML email templates and does not require a Salesforce.com username and password to view in an email (`true`) or not (`false`). If the images will be used as a custom app logo or custom tab icon, both of which require a Salesforce.com username and password to view, set this field to `false`. This field and `internalUseOnly` are mutually exclusive; you cannot set both to `true`.

Declarative Metadata Sample Definition

The following is the definition of a document :

```
<?xml version="1.0" encoding="UTF-8"?>
<Document xmlns="http://soap.sforce.com/2006/04/metadata">
    <internalUseOnly>false</internalUseOnly>
    <name>Q2 Campaign Analysis</name>
    <public>false</public>
    <description>Analyze Q2 campaign effectiveness</description>
</Document>
```

For a sample of using a document within a folder, see Folder on page 158.

See Also:

Folder

EmailTemplate

Represents an email template. This metadata type extends the MetadataWithContent component and shares its fields.

File Suffix and Directory Location

The file suffix is `.email` for the template file. The accompanying metadata file is named *`EmailTemplateName`*`-meta.xml`.

EmailTemplate components are stored in the `email` folder in the corresponding package directory.

Version

Email templates are available in API version 12.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`apiVersion`	double	The API version if this is a Visualforce email template. Every Visualforce email template has an API version specified at creation. This field is available in API version 16.0 and later.
`attachedDocuments`	string[]	A list of references to documents in your organization. These documents are included as attachments in the email template. Each document is referenced by its path, for example `MyFolder/MyDocument.txt`.
`attachments`	Attachment[]	A list of attachments for the email template.
`available`	boolean	Required. Indicates whether this template is offered to users when sending an email (`true`) or not (`false`).
`content`	base64Binary	Content of the email template. Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion is usually handled for you by a SOAP client. This field contains: • Binary content of the email body if `type` is set to `text` • HTML email content if `type` is set to `html` • HTML body if `type` is set to `custom` • Visualforce body if `type` is set to `visualforce` This field is inherited from the MetadataWithContent component.
`description`	string	The email template description. This can be useful to describe the reason for creating the template.
`encodingKey`	Encoding (enumeration of type string)	Required. The default encoding setting is Unicode: `UTF-8`. Change it if your template requires data in a different format. Valid values include: • `UTF-8` • `ISO-8859-1` • `Shift_JIS`

Field Name	Field Type	Description
		• ISO-2022-JP • EUC-JP • ks_c_5601-1987 • Big5 • GB2312
fullName	string	The email template developer name used as a unique identifier for API access. The fullName can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. If this field contained characters before version 14.0 that are no longer allowed, the characters were stripped out of this field, and the previous value of the field was saved in the name field. This field is inherited from the Metadata component.
letterhead	string	The letterhead name associated with this email template. Only available when type is set to html.
name	string	Required. Email template name. The list of characters allowed in the fullName field has been reduced for versions 14.0 and later. This field contains the value contained in the fullName field before version 14.0.
packageVersions	PackageVersion[]	The list of package versions for any managed packages containing components that are referenced by this email template. This field is only relevant for Visualforce email templates. For more information about managed packages, see the Force.com Quick Reference for Developing Packages. For more information about package versions, see "About Package Versions" in the Salesforce.com online help. This field is available in API version 16.0 and later.
style	EmailTemplateStyle (enumeration of type string)	Required. The style of the template. This field is only available when type is set to html. Valid style values include: • none • freeForm • formalLetter • promotionRight • promotionLeft • newsletter • products
subject	string	The email subject.
textOnly	string	The text of the email body if type is set to html or custom.
type	EmailTemplateType (enumeration of type string)	Required. The email template type. The valid values are: • text -all users can create or change text email templates. • html - administrators and users with the "Edit HTML Templates" permission can create HTML email templates based on a letterhead.

Field Name	Field Type	Description
		• `custom` - administrators and users with the "Edit HTML Templates" permission can create custom HTML email templates without using a letterhead. You must either know HTML or obtain the HTML code to insert in your email template. • `visualforce` - administrators and users with the "Customize Application" permission can create email templates using Visualforce.

Attachment

Attachment represents an email attachment.

Field	Field Type	Description
`content`	base64Binary	Required. The attachment content. Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion is usually handled for you by a SOAP client.
`name`	string	Required. The attachment file name.

Declarative Metadata Sample Definition

A sample XML definition of an template is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<EmailTemplate xmlns="http://soap.sforce.com/2006/04/metadata">
    <available>true</available>
    <description>Sample Email Template</description>
    <encodingKey>ISO-8859-1</encodingKey>
    <name>Sample Email Template</name>
    <style>none</style>
    <subject>Sample email subject</subject>
    <textOnly>Your case has been resolved.</textOnly>
    <type>custom</type>
</EmailTemplate>
```

See Also:

Letterhead

EntitlementTemplate

Represents an entitlement template. Entitlement templates are predefined terms of customer support that you can quickly add to products. For example, you can create entitlement templates for Web or phone support so that users can easily add entitlements to products offered to customers. EntitlementTemplate extends the Metadata metadata type and inherits its `fullName` field.

Declarative Metadata File Suffix and Directory Location

EntitlementTemplate components are stored in the `entitlementTemplates` directory of the corresponding package directory. The file name matches the unique name of the entitlement template, and the extension is `.entitlementTemplate`.

Version

Force.com EntitlementTemplate components are available in API version 18.0 and higher.

Fields

Field	Field Type	Description
`businessHours`	string	The entitlement's supported business hours.
`casesPerEntitlement`	int	Lets you limit the number of cases the entitlement supports.
`entitlementProcess`	string	The entitlement process associated with the entitlement.
`isPerIncident`	boolean	`true` if entitlements created from this template service a limited number of cases; `false` otherwise.
`term`	int	The number of days the entitlement is in effect.
`type`	string	The type of entitlement, such as Web or phone support.

Declarative Metadata Sample Definition

A sample XML definition of an entitlement template is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<EntitlementTemplate xmlns="http://soap.sforce.com/2006/04/metadata">
    <businessHours>Los Angeles Business Hours</businessHours>
    <casePackSize>5</casePackSize>
    <entitlementProcess>Gold Customer Support</entitlementProcess>
    <isCasePack>true</isCasePack>
    <term>5</term>
    <type>entitlement template type</type>
</EntitlementTemplate>
```

Folder

Represents a folder. It extends the Metadata metadata type and inherits its `fullName` field. Four folder types currently exist in application:

- Document folder
- Email template folder
- Report folder
- Dashboard folder

Note: If the value of `accessType` is `Shared`, granting access by group, role, or role and subordinates is not supported. For more information about granting access to records, see Granting Access to Records in the Salesforce.com online help.

File Suffix and Directory Location

Folders are stored in the `document` directory of the corresponding package directory. Folders do not have a text file representation--they are containers for documents. A metadata file accompanies each folder, named `*FolderName*.xls-meta.xml`.

Version

Folders are available in API version 11.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`accessType`	FolderAccessTypes (enumeration of type string)	Required. The type of access for this folder. Valid values are: • `Shared`. This folder is accessible only by the specified set of users. • `Public`. This folder is accessible by all users, including portal users. • `PublicInternal`. This folder is accessible by all users, excluding portal users. This setting is available for report and dashboard folders in organizations with a partner portal or Customer Portal enabled. • `Hidden`. This folder is hidden from all users.
`fullName`	string	The name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`name`	string	Required. The name of the document folder.
`publicFolderAccess`	PublicFolderAccess (enumeration of type string)	If `Public` is the value for `accessType`, this field indicates the type of access all users will have to the contents of the folder. Valid values include: • `ReadOnly`. All users can read the contents of the folder, but no user can change the contents. • `ReadWrite`. All users can read or change the contents of the folder.
`sharedTo`	SharedTo	Sharing access for the folder. See "Sharing Considerations" in the Salesforce.com online help.

Declarative Metadata Sample Definition

The following is the definition of a document folder that contains a document:

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <fullName>basic</fullName>
    <types>
        <members>sampleFolder</members>
        <members>sampleFolder/TestDocument.txt</members>
        <name>Document</name>
    </types>
    <version>20.0</version>
</Package>
```

See Also:

Document

HomePageComponent

Represents the metadata associated with a home page component. You can customize the Home tab to include components such as sidebar links, a company logo, or a dashboard snapshot. For more information, see "Customizing Home Tab Page Layouts" in the Salesforce.com online help. It extends the Metadata metadata type and inherits its `fullName` field. Use to create, update, or delete home page component definitions.

Declarative Metadata File Suffix and Directory Location

The file suffix for home page components is `.homePageComponent` and components are stored in the `homepagecomponents` directory of the corresponding package directory.

Version

Home page components are available in API version 12.0 and later.

HomePageComponent

This metadata type represents the valid values that define a home page component:

Field Name	Field Type	Description
`body`	string	If this is an HTML page component, this is the body of the HTML.
`fullName`	string	The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. Inherited from the Metadata component, this field is not defined in the WSDL for this component. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`links`	string[]	If the `pageComponentType` is `links`, then zero or more names of custom page links can be specified. • `ObjectWebLink` • `CustomPageWebLink`
`pageComponentType`	PageComponentType (enumeration of type string)	Required. Valid values are the following: • `links` • `htmlArea` • `imageOrNote`

Field Name	Field Type	Description
`width`	PageComponentWidth (enumeration of type string)	This field is only available for HTML components, and indicates whether this is a narrow or wide home page component. Valid values are the following: • `narrowComponents` • `wideComponents`

Declarative Metadata Sample Definition

The following is the definition of a home page component. See HomePageLayout on page 162 and Weblink on page 123 for related samples.

```
<?xml version="1.0" encoding="UTF-8"?>
<HomePageComponent xmlns="http://soap.sforce.com/2006/04/metadata">
    <width>wideComponents</width>
    <links>google</links>
    <pageComponentType>links</pageComponentType>
</HomePageComponent>
```

See Also:

HomePageLayout
Weblink

HomePageLayout

Represents the metadata associated with a home page layout. You can customize home page layouts and assign the layouts to users based on their user profile. For more information, see “Customizing Home Tab Page Layouts” in the Salesforce.com online help.

File Suffix and Directory Location

Home page layouts are stored in the `homePageLayouts` directory of the corresponding package directory. The extension is `.homePageLayout`.

Version

Home page components are available in API version 12.0 and later. It extends the Metadata metadata type and inherits its `fullName` field.

Fields

This metadata type represents the valid values that define a home page layout:

Field Name	Field Type	Description
fullName	string	The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. Inherited from the Metadata component, this field is not defined in the WSDL for this component. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
narrowComponents	string[]	The list of elements in the narrow column on the left side of the home page.
wideComponents	string[]	The list of elements in the wide column on the right side of the home page.

Declarative Metadata Sample Definition

The following is the definition of a home page layout. See HomePageComponent on page 161 and Weblink on page 123 for related samples.

```
<?xml version="1.0" encoding="UTF-8"?>
<HomePageLayout xmlns="http://soap.sforce.com/2006/04/metadata">
    <narrowComponents>google</narrowComponents>
</HomePageLayout>
```

See Also:

HomePageComponent
Weblink

Layout

Represents the metadata associated with a page layout. For more information, see "Managing Page Layouts" in the Salesforce.com online help. It extends the Metadata metadata type and inherits its `fullName` field.

Note: If you want to edit the Ideas layout, you must specify it by name in the `package.xml` file. In `package.xml`, use the following code to retrieve the Ideas layout:

```
<types>
    <members>Idea-Idea Layout</members>
    <name>Layout</name>
</types>
```

File Suffix and Directory Location

Layouts are stored in the `layouts` directory of the corresponding package directory. The extension is `.layout`.

Note: Retrieving a component of this metadata type in a project makes the component appear in the Profile component as well.

Version

Layouts are available in API version 13.0 and later.

Fields

This metadata type represents the valid values that define a page layout:

Field Name	Field Type	Description
`customButtons`	string[]	The custom buttons for this layout. Each button is a reference to a Weblink on the same object. For example, a ButtonLink refers to a Weblink on the same standard or custom object named 'ButtonLink'.
`emailDefault`	boolean	Only relevant if `showEmailCheckbox` is set; indicates the default value of that checkbox.
`excludeButtons`	string[]	List of standard buttons to exclude from this layout. For example, `<excludeButtons>Delete</excludeButtons>`, will exclude the "Delete" button from this layout.
`fullName`	string	If this is an HTML page component, this is the body of the HTML. For case close layouts you must prefix the component name with `CaseClose-`. For example, a layout named `MyLayout` must be named `CaseClose-MyLayout` if it is a case close layout.
`headers`	LayoutHeader[] (enumeration of type string)	Layout headers are currently only used for tagging, and only appear in the UI if tagging is enabled. For more information, see "About Tagging" in the Salesforce.com online help. Valid string values are: • `PersonalTagging`—tag is private to user. • `PublicTagging`—tag can be viewed by any other user who can access the record.
`layoutSections`	LayoutSection[]	The main sections of the layout containing fields, s-controls, and custom links. The order here determines the layout order.
`miniLayout`	MiniLayout	A mini layout is used in the mini view of a record in the console, hover details, and event overlays.
`multilineLayoutFields`	string[]	Fields for the special multiline layout fields which appear in OpportunityProduct layouts. These are otherwise similar to `miniLayoutFields` `miniLayout`.
`relatedLists`	RelatedListItem[]	The related lists for the layout, listed in the order they appear in the UI..
`relatedObjects`	string[]	The list of related objects that appears in the mini view of the console. In database terms, these are foreign key fields on the object for the layout. For more information, see "Choosing Related Objects for the Console's Mini View" in the Salesforce.com online help.
`runAssignmentRulesDefault`	boolean	Only relevant if `showRunAssignmentRulesCheckbox` is set; indicates the default value of that checkbox.

Field Name	Field Type	Description
`showEmailCheckbox`	boolean	Only allowed on Case, CaseClose, and Task layouts. If set, a checkbox appears to show email.
`showRunAssignmentRulesCheckbox`	boolean	Only allowed on Lead and Case objects. If set, a checkbox appears on the page to show assignment rules.
`showSolutionSection`	boolean	Only allowed on CaseClose layout. If set, the built-in solution information section shows up on the page.
`showSubmitAndAttachButton`	boolean	Only allowed on Case layout. If set, the **Submit & Add Attachment** button displays on case edit pages to portal users in the Customer Portal.

MiniLayout

A mini layout is used in the mini view of a record in the console, hover details, and event overlays.

Field Name	Field Type	Description
`fields`	string[]	The fields for the mini-layout, listed in the order they appear in the UI. Fields that appear here must appear in the main layout.
`relatedLists`	RelatedListItem[]	The mini related list, listed in the order they appear in the UI. You cannot set sorting on mini related lists. Fields that appear here must appear in the main layout.

LayoutSection

LayoutSection represents a section of a page layout, such as the Custom Links section.

Field Name	Field Type	Description
`customLabel`	boolean	Indicates if this section's label is custom or standard (built-in). Custom labels can be any text, but must be translated. Standard labels have a predefined set of valid values, for example 'System Information', which are automatically translated.
`detailHeading`	boolean	Controls if this section appears in the detail page. In the UI, this setting corresponds to the checkbox in the section details dialog.
`editHeading`	boolean	Controls if this section appears in the edit page.
`label`	string	The label; either standard or custom, based on the `customLabel` flag.
`layoutColumns`	LayoutColumn[]	The columns of the layout, depending on the style. There may be 1, 2, or 3 columns, ordered left to right.
`style`	LayoutSectionStyle (enumeration of type string)	The style of the layout: • `TwoColumnsTopToBottom` - Two columns, tab goes top to bottom • `TwoColumnsLeftToRight` - Two columns, tab goes left to right • `OneColumn` - One column • `CustomLinks` - Contains custom links only

Field Name	Field Type	Description
`summaryLayout`	SummaryLayout on page 166	Reserved for future use.

LayoutColumn

LayoutColumn represents the items in a column within a layout section.

Field Name	Field Type	Description
`layoutItems`	LayoutItem[]	The individual items within a column (ordered from top to bottom).
`reserved`	string	This field is reserved for Salesforce.com. The field resolves an issue with some SOAP libraries. Any value entered in the field is ignored.

LayoutItem

LayoutItem represents the valid values that define a layout item. An item must have only one of the following set: customLink, field, scontrol, page.

Field Name	Field Type	Description
`behavior`	UiBehavior (enumeration of type string)	Determines the field behavior. Valid string values: • `Edit` - The layout field can be edited but is not required • `Required` - The layout field can be edited and is required • `Readonly` - The layout field is read-only
`customLink`	string	The `customLink` reference. This is only allowed inside a `CustomLink layoutSection`.
`emptySpace`	boolean	Controls if this layout item is a blank space.
`field`	string	The field name reference, relative to the layout object, for example `Description` or `MyField__c`.
`height`	int	For s-control and pages only, the height in pixels.
`page`	string	Reference to a Visualforce page.
`scontrol`	string	Reference to an s-control.
`showLabel`	boolean	For s-control and pages only, whether or not to show the label.
`showScrollbars`	boolean	For s-control and pages only, whether or not to show scrollbars.
`width`	string	For s-control and pages only, the width in pixels or percent. Pixel values are simply the number of pixels, for example, `500`. Percentage values must include the percent sign, for example, `20%`.

RelatedListItem

RelatedListItem represents a related list in a page layout.

Field Name	Field Type	Description
customButtons	string[]	A list of custom buttons used in the related list. For more information, see "Defining Custom Buttons and Links" in the Salesforce.com online help.
excludeButtons	string[]	A list of excluded related-list buttons.
fields	string[]	A list of fields displayed in the related list.
relatedList	string	Required. The name of the related list.
sortField	string	The name of the field that is used for sorting.
sortOrder	SortOrder (enumeration of type string)	If the sortField is set, the sortOrder field determines the sort order. • Asc - sort in ascending order • Desc - sort in descending order

SummaryLayout

Reserved for future use.

SummaryLayoutItem

Reserved for future use.

Declarative Metadata Sample Definition

The following is the definition of a page layout:

```
<?xml version="1.0" encoding="UTF-8"?>
<Layout xmlns="http://soap.sforce.com/2006/04/metadata">
    <customButtons>ButtonLink</customButtons>
    <layoutSections>
        <editHeading>true</editHeading>
        <label>Information</label>
        <layoutColumns>
            <layoutItems>
                <behavior>Required</behavior>
                <field>Name</field>
            </layoutItems>
            <layoutItems>
                <height>180</height>
                <scontrol>LayoutSControl</scontrol>
                <showLabel>true</showLabel>
                <showScrollbars>true</showScrollbars>
                <width>50%</width>
            </layoutItems>
        </layoutColumns>
        <layoutColumns>
            <layoutItems>
                <behavior>Edit</behavior>
                <field>OwnerId</field>
            </layoutItems>
            <layoutItems>
                <behavior>Edit</behavior>
                <field>CurrencyIsoCode</field>
            </layoutItems>
        </layoutColumns>
        <style>TwoColumnsTopToBottom</style>
    </layoutSections>
    <layoutSections>
        <editHeading>true</editHeading>
        <label>System Information</label>
        <layoutColumns>
            <layoutItems>
                <behavior>Readonly</behavior>
                <field>CreatedById</field>
            </layoutItems>
            <layoutItems>
                <behavior>Readonly</behavior>
                <field>Alpha1__c</field>
            </layoutItems>
        </layoutColumns>
        <layoutColumns>
            <layoutItems>
                <behavior>Readonly</behavior>
                <field>LastModifiedById</field>
            </layoutItems>
            <layoutItems>
                <behavior>Edit</behavior>
                <field>TextArea__c</field>
            </layoutItems>
        </layoutColumns>
        <style>TwoColumnsTopToBottom</style>
    </layoutSections>
    <layoutSections>
        <customLabel>true</customLabel>
        <detailHeading>true</detailHeading>
        <label>Custom Links</label>
        <layoutColumns>
            <layoutItems>
                <customLink>CustomWebLink</customLink>
            </layoutItems>
        </layoutColumns>
        <style>CustomLinks</style>
    </layoutSections>
    <miniLayoutFields>Name</miniLayoutFields>
    <miniLayoutFields>OwnerId</miniLayoutFields>
    <miniLayoutFields>CurrencyIsoCode</miniLayoutFields>
    <miniLayoutFields>Alpha1__c</miniLayoutFields>
    <miniLayoutFields>TextArea__c</miniLayoutFields>
    <miniRelatedLists>
        <relatedList>RelatedNoteList</relatedList>
    </miniRelatedLists>
    <relatedLists>
        <fields>StepStatus</fields>
        <fields>CreatedDate</fields>
        <fields>OriginalActor</fields>
        <fields>Actor</fields>
        <fields>Comments</fields>
        <fields>Actor.Alias</fields>
        <fields>OriginalActor.Alias</fields>
        <relatedList>RelatedProcessHistoryList</relatedList>
    </relatedLists>
    <relatedLists>
        <relatedList>RelatedNoteList</relatedList>
    </relatedLists>
</Layout>
```

Letterhead

Represents formatting options for the letterhead in an email template. Letterheads define the look and feel of your HTML email templates. Your HTML email templates can inherit the logo, color, and text settings from a letterhead. For more information, see "Creating Letterheads" in the Salesforce.com online help. It extends the Metadata metadata type and inherits its `fullName` field.

File Suffix and Directory Location

The file suffix for letterheads is `.letter` and components are stored in the `letterhead` directory of the corresponding package directory.

Version

Letterheads are available in API version 12.0 and later.

Fields

With the exception of logo, and horizontal and vertical alignment, all of these fields are required.

Field Name	Field Type	Description
`available`	boolean	Required. Indicates whether this letterhead can be used (`true`) or not (`false`), for example, in an email template.
`backgroundColor`	string	Required. The background color, in hexadecimal, for example `#FF6600`.
`bodyColor`	string	Required. The body color in hexadecimal.
`bottomLine`	LetterheadLine (enumeration of type string)	Required. The style for the bottom line. Valid style values include: • `color`. The color of the line in hexadecimal, as a string value. • `height`. The height of the line, as an int value.
`description`	string	Text description of how this letterhead differs from other letterheads.
`fullName`	string	The internal name of the letterhead, based on the `name`, but with white spaces and special characters escaped out for validity.
`footer`	LetterheadHeaderFooter	Required. The style for the footer.
`header`	LetterheadHeaderFooter	Required. The style for the header.

Field Name	Field Type	Description
`middleLine`	LetterheadLine	Required. The style for the middle border line in your letterhead. Valid style values include: • `color`. The color of the line in hexadecimal, as a string value. • `height`. The height of the line, as an int value.
`name`	string	Required. The name of the letterhead.
`topLine`	LetterheadLine	Required. The style for the top horizontal line below the header. Valid style values include: • `color`. The color of the line in hexadecimal, as a string value. • `height`. The height of the line, as an int value.

LetterheadHeaderFooter

LetterheadHeaderFooter represents the properties of a header or footer.

Field	Field Type	Description
`backgroundColor`	string	Required. The background color of the header or footer in hexadecimal format.
`height`	DashboardComponent[]	Required. The height of the header or footer.
`horizontalAlignment`	LetterheadHorizontalAlignment (enumeration of type string)	The horizontal alignment of the header or footer. Valid values are: • `None` • `Left` • `Center` • `Right`
`logo`	string	The logo which is a reference to a document, for example `MyFolder/MyDocument.gif`.
`verticalAlignment`	LetterheadVerticalAlignment (enumeration of type string)	The vertical alignment of the header or footer. Valid values are: • `None` • `Top` • `Middle` • `Bottom`

LetterheadLine

LetterheadLine represents the properties of a line.

Field	Field Type	Description
`color`	string	Required. The color of the line in hexadecimal format.
`height`	int	Required. The height of the line.

Declarative Metadata Sample Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<Letterhead xmlns="http://soap.sforce.com/2006/04/metadata">
    <available>true</available>
    <backgroundColor>#CCCCCC</backgroundColor>
    <bodyColor>#33FF33</bodyColor>
    <bottomLine>
        <color>#3333FF</color>
        <height>5</height>
    </bottomLine>
    <description>INITIAL</description>
    <footer>
        <backgroundColor>#FFFFFF</backgroundColor>
        <height>100</height>
        <horizontalAlignment>Left</horizontalAlignment>
        <verticalAlignment>Top</verticalAlignment>
    </footer>
    <header>
        <backgroundColor>#FFFFFF</backgroundColor>
        <height>100</height>
        <horizontalAlignment>Left</horizontalAlignment>
        <verticalAlignment>Top</verticalAlignment>
    </header>
    <middleLine>
        <color>#AAAAFF</color>
        <height>5</height>
    </middleLine>
    <name>SimpleLetterheadLabel</name>
    <topLine>
        <color>#FF99FF</color>
        <height>5</height>
    </topLine>
</Letterhead>
```

Metadata

This is the base class for all metadata types. You cannot edit this object. A component is an instance of a metadata type.

Metadata is analogous to sObject, which represents all standard objects. Metadata represents all components and fields in the Metadata API. Instead of identifying each component with an ID, each custom object or custom field has a unique `fullName`, which must be distinct from standard object names, as it must be when you create custom objects or custom fields in the Salesforce.com user interface.

Version

Metadata components are available in API version 10.0 and later.

Fields

Field Name	Field Type	Description
`fullName`	string	Required. The name of the component. If a field, the name must specify the parent object, for example `Account.FirstName`. The __c suffix must be appended to custom object names and custom field names when you are setting the `fullName`. For example, a custom field in a custom object could have a `fullName` of `MyCustomObject__c.MyCustomField__c`.

See Also:

CustomObject
CustomField
MetadataWithContent

MetadataWithContent

This is the base type for all metadata types that contain content, such as documents or email templates. It extends Metadata. You cannot edit this object.

Version

MetadataWithContent components are available in API version 14.0 and later.

Fields

Field Name	Field Type	Description
`content`	base64Binary	Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion is usually handled for you by a SOAP client.
`fullName`	string	Required. The name of the component. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. Inherited from the Metadata component, this field is not defined in the WSDL for this component. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.

See Also:

Metadata

Portal

The Portal metadata type represents a partner portal or Customer Portal, and extends Metadata and inherits its `fullName` field. To use this metadata type, you must have a partner portal or Customer Portal enabled for your organization. For more information, see "Partner Portal Overview" and "Enabling Your Customer Portal" in the Salesforce.com online help.

Declarative Metadata File Suffix and Directory Location

Force.com Portal components are stored in the `portals` directory of the corresponding package directory. The file name matches the portal name, and the extension is `.portal`.

Version

Force.com Portal components are available in API version 15.0 and later.

Fields

Field	Field Type	Description
`active`	boolean	Required. Denotes whether this portal is active.
`admin`	string	The full name of the user designated to administer the portal.
`defaultLanguage`	string	The default language for HTML messages for the portal. Use the abbreviation for the language, for example, en_US for United States English.
`description`	string	The portal description.
`emailSenderAddress`	string	Required. The email address used when sending emails using templates configured from the portal (for example, for resetting the password).
`emailSenderName`	string	Required. The name to display when sending emails using templates configured from the portal (for example, for resetting the password).
`enableSelfCloseCase`	boolean	For the Customer Portal, allows portal users to close their own cases.
`footerDocument`	string	The file to be used as the footer for this portal.
`forgotPassTemplate`	string	The email template to use when a user clicks the **Forgot Password** link.
`fullName`	string	Required. The name of the portal. Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`headerDocument`	string	The file to be used as the header for this portal.
`isSelfRegistrationActivated`	boolean	Determines whether self-registration is active or not for this portal.

Field	Field Type	Description
loginHeaderDocument	string	The file to be used as the header for this portal's login page.
logoDocument	string	The file to be used as the logo for this portal.
logoutUrl	string	The URL that the user should be redirected to on logout.
newCommentTemplate	string	The email template to be used for auto-notifications on new case comments.
newPassTemplate	string	The email template to be used for auto-notifications on password reset.
newUserTemplate	string	The email template to be used for auto-notifications on new user creation.
ownerNotifyTemplate	string	The email template to be used for auto-notifications on owner change.
selfRegNewUserUrl	string	The URL of the self-registration page.
selfRegUserDefaultProfile	string	The default profile for self-registered users.
selfRegUserDefaultRole	PortalRoles (enumeration of type string)	The default role for self-registered users. The valid values are: • Executive • Manager • User • PersonAccount
selfRegUserTemplate	string	The email template to be used for auto-notifications on self-registration.
showActionConfirmation	boolean	Determines whether or not confirmation messages are displayed for actions in the portal.
stylesheetDocument	string	The Document object to be used as the CSS stylesheet for this portal.
type	PortalType (enumeration of type string)	Required. The type for this portal. The valid values are: • CustomerSuccess • Partner

Declarative Metadata Sample Definition

A sample XML definition of a portal is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Portal xmlns="http://soap.sforce.com/2006/04/metadata">
    <active>true</active>
    <description>Customer Portal</description>
    <emailSenderName>rguest@albany.com</emailSenderName>
    <enableSelfCloseCase>false</enableSelfCloseCase>
    <forgotPassTemplate>unfiled$public/ChangePwdEmail</forgotPassTemplate>
    <isSelfRegistrationActivated>false</isSelfRegistrationActivated>
    <newPassTemplate>unfiled$public/ChangePwdEmail</newPassTemplate>
    <newUserTemplate>unfiled$public/NewUserEmail</newUserTemplate>
    <selfRegUserTemplate>unfiled$public/SelfRegUserEmail</selfRegUserTemplate>
    <showActionConfirmation>false</showActionConfirmation>
    <type>CustomerSuccess</type>
</Portal>
```

See Also:

CustomSite

Profile

Represents a user profile. A profile defines a user's permission to perform different functions within Salesforce.com. For more information, see “Managing Profiles” in the Salesforce.com online help. It extends the Metadata metadata type and inherits its `fullName` field.

Declarative Metadata File Suffix and Directory Location

The file suffix is `.profile`. There is one file for each profile, stored in the `profiles` folder in the corresponding package directory.

Version

Profiles are available in API version 10.0 and later.

Fields

The contents of a profile returned by the Metadata API depends on the contents requested in the RetrieveRequest message. For example, profiles only include field-level security for fields included in custom objects returned in the same RetrieveRequest as the profiles. The profile definition contains the following fields:

Field Name	Field Type	Description
`applicationVisibilities`	ProfileApplicationVisibility[]	Indicates which applications are visible to users assigned to this profile.
`classAccesses`	ProfileApexClassAccess[]	Indicates which top-level Apex classes have methods that users assigned to this profile can execute.
`fieldLevelSecurities`	ProfileFieldLevelSecurity[]	Indicates which fields are visible to a user assigned to this profile.
`fullName`	string	The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. Inherited from the Metadata component, this field is not defined in the WSDL for this component. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`layoutAssignments`	ProfileLayoutAssignments[]	Indicates which layout to use for this profile.
`loginIpRanges`	ProfileLoginIpRange[]	The list of IP address ranges from which users with a particular profile can log in. This field is available in API version 17.0 and later.

Field Name	Field Type	Description
`objectPermissions`	ProfileObjectPermissions[]	Indicates which objects are accessible to a user assigned to this profile, and the kind of access available (create, read, update, delete).
`pageAccesses`	ProfileApexPageAccess[]	Indicates which custom Visualforce controllers or controller extensions applied to a Visualforce page have methods that users assigned to this profile can execute.
`recordTypeVisibilities`	ProfileRecordTypeVisibility[]	Indicates the visibility of record types for users assigned to this profile.
`tabVisibilities`	ProfileTabVisibility[]	Indicates which record types are visible to a user assigned to this profile, and therefore which tabs within an application are visible.
`userLicense`	string	The `User License` for the profile. A user license entitles a user to different functionality within Salesforce.com and determines the profiles available to the user. This field is available in API version 17.0 and later.

ProfileApplicationVisibility

ProfileApplicationVisibility determines if an application is visible to a user assigned to this profile:

Field Name	Field Type	Description
`application`	string	Required. The name of the application.
`default`	boolean	Required. Indicates whether the application is the default application (`true`) or not (`false`). Only one application per profile can be set to `true`.
`visible`	boolean	Required. Indicates whether this application is visible to users assigned to this profile (`true`) or not (`false`).

ProfileApexClassAccess

ProfileApexClassAccess determines which top-level Apex classes have methods that users assigned to this profile can execute. For more information, see "Setting Apex Class Security" in the Salesforce.com online help.

Field Name	Field Type	Description
`apexClass`	string	Required. The Apex class name.
`enabled`	boolean	Required. Indicates whether users assigned to this profile can execute methods in the top-level class (`true`) or not (`false`).

ProfileFieldLevelSecurity

ProfileFieldLevelSecurity represents the field level security for users assigned to a profile:

Field Name	Field Type	Description
`editable`	boolean	Required. Indicates whether this field is editable (`true`) or not (`false`).

Field Name	Field Type	Description
`field`	string	Required. Indicates the name of the field.
`hidden`	boolean	Indicates whether this field is hidden (`true`) or not (`false`). For portal profiles, this is set to `true` by default in API version 19.0 and later.

ProfileLayoutAssignments

ProfileLayoutAssignments determines which layout to use for a profile and a given entity:

Field Name	Field Type	Description
`layout`	string	Required. Indicates the layout for this particular entity.
`recordType`	string	This field is optional. If the `recordType` of the record matches a layout assignment rule, it will use the specified layout.

ProfileLoginIpRange

ProfileLoginIpRange IP defines an IP address ranges from which users with a particular profile can log in. See “Restricting Login IP Ranges on Profiles” in the Salesforce.com online help.

Field Name	Field Type	Description
`endAddress`	string	Required. The end IP address for the range.
`startAddress`	string	Required. The start IP address for the range.

ProfileObjectPermissions

ProfileObjectPermissions represents a user's access to objects.

Note: In API version 18.0 and later, these permissions are disabled in new custom objects for any profiles in which “View All Data” or “Modify All Data” is disabled.

Field Name	Field Type	Description
`allowCreate`	boolean	Indicates whether the object referenced by the `object` field can be created by the users assigned to this profile (`true`) or not (`false`). This field is named `revokeCreate` before version 14.0 and the logic is reversed. The field name change and the update from `true` to `false` and vice versa is automatically handled between versions and does not require any manual editing of existing XML component files. The field name change and the update from `true` to `false` and vice versa is automatically handled between versions and does not require any manual editing of existing XML component files.

Field Name	Field Type	Description
allowDelete	boolean	Indicates whether the object referenced by the `object` field can be deleted by the users assigned to this profile (`true`) or not (`false`). This field is named `revokeDelete` before version 14.0 and the logic is reversed. The field name change and the update from `true` to `false` and vice versa is automatically handled between versions and does not require any manual editing of existing XML component files.
allowEdit	boolean	Indicates whether the object referenced by the `object` field can be edited by the users assigned to this profile (`true`) or not (`false`). This field is named `revokeEdit` before version 14.0 and the logic is reversed. The field name change and the update from `true` to `false` and vice versa is automatically handled between versions and does not require any manual editing of existing XML component files.
allowRead	boolean	Indicates whether the object referenced by the `object` field can be seen by the users assigned to this profile (`true`) or not (`false`). This field is named `revokeRead` before version 14.0 and the logic is reversed. The field name change and the update from `true` to `false` and vice versa is automatically handled between versions and does not require any manual editing of existing XML component files.
modifyAllRecords	boolean	Indicates whether the object referenced by the `object` field can be read, edited, or deleted by the users assigned to this profile (`true`) or not (`false`), regardless of the sharing settings for the object. This is equivalent to the “Modify All Data” user permission limited to the individual object level. This is a new field in API version 15.0. **Note:** This field is not available for all objects. Refer to the profile in the user interface to determine which objects currently support these permissions. Profiles with "Modify All Data" ignore `modifyAllRecords` entries in the Metadata API and don't return an error if "Modify All Data" is enabled on the profile.
object	string	Required. The name of the object whose permissions are altered by this profile, for example, `MyCustomObject__c`.
viewAllRecords	boolean	Indicates whether the object referenced by the `object` field can be read by the users assigned to this profile (`true`) or not (`false`), regardless of the sharing settings for the object. This includes private records (records with no parent object). This is equivalent to the “View All Data” user permission limited to the individual object level. This is a new field in API version 15.0. **Note:** This field is not available for all objects. Refer to the profile in the user interface to determine which objects currently support these permissions. Profiles with "View All

Field Name	Field Type	Description
		Data" ignore `viewAllRecords` entries in the Metadata API and don't return an error if "View All Data" is enabled on the profile.

ProfileApexPageAccess

ProfileApexPageAccess determines which custom Visualforce controllers or controller extensions applied to a Visualforce page have methods that users assigned to this profile can execute. For more information, see “Setting Apex Class Security” in the Salesforce.com online help.

Field Name	Field Type	Description
`apexPage`	string	Required. The Visualforce page name.
`enabled`	boolean	Required. Indicates whether users assigned to this profile can execute methods in the custom controller or controller extension applied to the page (`true`) or not (`false`).

ProfileRecordTypeVisibility

ProfileRecordTypeVisibility represents the visibility of record types for this profile. Record types allow you to offer different business processes, picklist values, and page layouts to different users based on their profiles. For more information, see “Managing Record Types” in the Salesforce.com online help.

Field Name	Field Type	Description
`default`	boolean	Required. Indicates whether the record type is the default for this pair of profile and object (`true`) or not (`false`). Only one default is allowed per object.
`personAccountDefault`	boolean	Indicates whether the record type is the default person account record type for this pair of profile and object (`true`) or not (`false`). Only one person account record type default is allowed per object. This field is only relevant for record types for account or contact objects. A person account is an individual consumer with whom you do business, such as a financial services client, an online shopper, or a vacation traveler. Person accounts are applicable to organizations that operate on a business-to-consumer model as opposed to a business-to-business model. For more information about person accounts, see “What is a Person Account?” in the Salesforce.com online help. Person accounts are not enabled by default in Salesforce.com. To request person accounts, contact salesforce.com.
`recordType`	string	Required. The record type name, for example `Account.MyRecordType`.
`visible`	boolean	Required. Indicates whether this record type is visible to users assigned to this profile (`true`) or not (`false`).

ProfileTabVisibility

ProfileTabVisibility represents the visibility of tabs for this profile. For version 17.0 and later, ProfileTabVisibility supports visibility of tabs for standard objects. The manifest file must include the standard object corresponding to a standard tab to retrieve the tab visibility in a profile.

Field Name	Field Type	Description
`tab`	string	Required. The name of the tab.
`visibility`	TabVisibility (enumeration of type string)	Required. Indicates the visibility of the tab. Valid values are: • `Hidden` • `DefaultOff` • `DefaultOn` For more information, see “Setting Tab Visibility” in the Salesforce.com online help.

Java Sample

The following sample uses picklists, profiles, and record types:

```
private void profileSample()
    throws Exception
{
    // Create an expense report record, tab and application...
    CustomObject expenseRecord = new CustomObject();
    expenseRecord.setFullName("ExpenseReport__c");
    expenseRecord.setLabel("Expense Report");
    expenseRecord.setPluralLabel("Expense Reports");

    expenseRecord.setDeploymentStatus(DeploymentStatus.Deployed);
    expenseRecord.setSharingModel(SharingModel.ReadWrite);

    CustomField nameField = new CustomField();
    nameField.setType(FieldType.AutoNumber);
    nameField.setLabel("Expense Report Number");
    nameField.setDisplayFormat("ER-{0000}");
    expenseRecord.setNameField(nameField);

    AsyncResult[] arsExpenseRecord =
        metadataBinding.create(expenseRecord);

    Picklist expenseStatus = new Picklist();
    PicklistValue unsubmitted = new PicklistValue();
    unsubmitted.setFullName("Unsubmitted");
    PicklistValue submitted = new PicklistValue();
    submitted.setFullName("Submitted");
    PicklistValue approved = new PicklistValue();
    approved.setFullName("Approved");
    PicklistValue rejected = new PicklistValue();
    rejected.setFullName("Rejected");
    expenseStatus.setPicklistValues(new PicklistValue[] {
        unsubmitted, submitted, approved, rejected}
    );

    CustomField expenseStatusField = new CustomField();
    expenseStatusField.setFullName("ExpenseReport__c.ExpenseStatus__c");
    expenseStatusField.setLabel("Expense Report Status");
    expenseStatusField.setType(FieldType.Picklist);
    expenseStatusField.setPicklist(expenseStatus);
    AsyncResult[] arsStatusField =
        metadataBinding.create(expenseStatusField);

    CustomTab expenseTab = new CustomTab();
    expenseTab.setFullName("ExpenseReport__c");
    expenseTab.setMotif("Custom70: Handsaw");
    expenseTab.setCustomObject(true);
    AsyncResult[] arsTab =
        metadataBinding.create(expenseTab);

    CustomApplication application = new CustomApplication();
    application.setFullName("ExpenseForce");
    application.setTab(new String[] {expenseTab.getFullName()});
    AsyncResult[] arsApp =
        metadataBinding.create(application);

    // Employees and managers have the same app visibility...
    ProfileApplicationVisibility appVisibility =
        new ProfileApplicationVisibility();
    appVisibility.setApplication("ExpenseForce");
    appVisibility.setVisible(true);

    Profile employee = new Profile();
    employee.setFullName("Employee");
    employee.setApplicationVisibilities(
        new ProfileApplicationVisibility[] {appVisibility}
    );
    AsyncResult[] arsProfileEmp =
        metadataBinding.create(employee);

    Profile manager = new Profile();
    manager.setFullName("Manager");
    manager.setApplicationVisibilities(
        new ProfileApplicationVisibility[] {appVisibility}
    );
    AsyncResult[] arsProfileMgr =
        metadataBinding.create(manager);

    // But employees and managers have different access
    // to the state of the expense sheet
    RecordType edit = new RecordType();
    edit.setFullName("ExpenseReport__c.Edit");
    RecordTypePicklistValue editStatuses = new RecordTypePicklistValue();
    editStatuses.setPicklist("ExpenseStatus__c");
    editStatuses.setValues(new PicklistValue[] {unsubmitted, submitted});
    edit.setPicklistValues(new RecordTypePicklistValue[] {editStatuses});
    AsyncResult[] arsRecTypeEdit =
        metadataBinding.create(edit);

    RecordType approve = new RecordType();
    approve.setFullName("ExpenseReport__c.Approve");
    RecordTypePicklistValue approveStatuses = new RecordTypePicklistValue();
    approveStatuses.setPicklist("ExpenseStatus__c");
    approveStatuses.setValues(new PicklistValue[] {approved, rejected});
    approve.setPicklistValues(new RecordTypePicklistValue[] {approveStatuses});
    AsyncResult[] arsRecTypeApp =
        metadataBinding.create(approve);
}
```

Declarative Metadata Sample Definition

The following is the definition of the standard profile in an organization with one custom object, `TestWeblinks__c` and one record type, `My First Recordtype`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile xmlns="http://soap.sforce.com/2006/04/metadata">
    <applicationVisibilities>
```

```
        <application>Myriad Publishing</application>
        <default>false</default>
        <visible>true</visible>
    </applicationVisibilities>
    <objectPermissions>
        <object>TestWeblinks__c</object>
    </objectPermissions>
    <recordTypeVisibilities>
        <default>true</default>
        <recordType>TestWeblinks__c.My First Recordtype</recordType>
        <visible>true</visible>
    </recordTypeVisibilities>
    <tabVisibilities>
        <tab>Myriad Publications</tab>
        <visibility>DefaultOn</visibility>
    </tabVisibilities>
</Profile>
```

Usage

When you use the `retrieve()` call to get information about profiles in your organization, the returned `.profile` files only include security settings for the other metadata types referenced in the retrieve request. For example, the `package.xml` file below contains a `types` element that matches all custom objects, so the returned profiles contain object and field permissions for all custom objects in your organization, but do not include permissions for standard objects, such as Account, and standard fields.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>CustomObject</name>
    </types>
   <types>
        <members>*</members>
        <name>Profile</name>
    </types>
    <version>20.0</version>
</Package>
```

The wildcard "*" on CustomObject does not match standard objects and this helps to avoid making unintended, high-impact profile changes. If you create a few custom objects in a Developer Edition organization, `retrieve()` the information, and subsequently `deploy()` the custom objects to your production organization, the profile and field-level security for all your standard objects, such as Account, and standard fields are not overwritten unless you explicitly create separate `types` elements for the standard objects or fields. For more information about profiles, see "Managing Profiles" in the Salesforce.com online help.

The Metadata API intentionally makes it somewhat difficult to include standard fields in `retrieve()` calls in order to prevent unexpected profile changes. However, you can still retrieve and deploy profile permissions for custom and standard fields in standard objects, such as Account.

The next `package.xml` file allows you to return profile permissions for Account standard and custom fields. Note how the standard Account object is defined in a `types` element by specifying it as a member of a CustomObject type.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>Account</members>
        <name>CustomObject</name>
    </types>
   <types>
        <members>*</members>
        <name>Profile</name>
    </types>
    <version>20.0</version>
</Package>
```

The final `package.xml` file allows you to return profile permissions for the `MyCustomField__c` custom field in the Account object.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
   <types>
        <members>Account.MyCustomField__c</members>
        <name>CustomField</name>
   </types>
   <types>
        <members>*</members>
        <name>Profile</name>
   </types>
   <version>20.0</version>
</Package>
```

RemoteSiteSetting

Represents a remote site setting. Before any Visualforce page, Apex callout, or JavaScript code using XmlHttpRequest in an s-control or custom button can call an external site, that site must be registered in the Remote Site Settings page, or the call will fail. RemoteSiteSetting extends the Metadata metadata type and inherits its `fullName` field.

Declarative Metadata File Suffix and Directory Location

RemoteSiteSetting components are stored in the `remoteSiteSettings` directory of the corresponding package directory. The file name matches the unique name of the remote site setting, and the extension is `.remoteSite`.

Version

RemoteSiteSetting components are available in API version 19.0 and later.

Fields

Field	Field Type	Description
`description`	string	The description explaining what this remote site setting is used for.
`disableProtocolSecurity`	boolean	Required. Indicates whether code within Salesforce.com can access the remote site regardless of whether the user's connection is over HTTP or HTTPS (`true`) or not (`false`). When `true`, code within Salesforce.com can pass data from an HTTPS session to an HTTP session, and vice versa. **Caution:** Only set to `true` if you understand the security implications.
`fullName`	string	The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot

Field	Field Type	Description
		end with an underscore or contain two consecutive underscore characters. Inherited from the Metadata component, this field is not defined in the WSDL for this component. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`isActive`	boolean	Required. Indicates if the remote site setting is active (`true`) or not (`false`).
`url`	string	Required. The URL for the remote site.

Declarative Metadata Sample Definition

A sample XML definition of a remote site setting is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<RemoteSiteSetting xmlns="http://soap.sforce.com/2006/04/metadata">
    <description>Used for Apex callout to mapping web service</description>
    <disableProtocolSecurity>false</disableProtocolSecurity>
    <isActive>true</isActive>
    <url>https://www.maptestsite.net/mapping1</url>
</RemoteSiteSetting>
```

Report

Represents a custom report. It extends the Metadata metadata type and inherits its `fullName` field. This metadata type only supports custom reports; standard reports are not supported. For more information, see “Reports Overview” in the Salesforce.com online help.

Declarative Metadata File Suffix and Directory Location

Reports are stored in the `reports` directory of the corresponding package directory. The file name matches the report title and the extension is `.report`.

Version

Report components are available in API version 14.0 and later.

Fields

The following information assumes that you are familiar with creating and running reports. For more information on these fields, see “Creating a Custom Report” in the Salesforce.com online help.

Field	Field Type	Description
`aggregates`	ReportAggregate[]	List that defines custom summary formulas for summary and matrix reports.

Field	Field Type	Description
`chart`	ReportChart	Defines a chart for summary and matrix reports .
`colorRanges`	ReportColorRange[]	List that specifies conditional highlighting for report summary data.
`columns`	ReportColumn[]	List that specifies the fields displayed in the report. Fields appear in the report in the same order as they appear in the Metadata API file.
`currency`	`CurrencyIsoCode` (enumeration of type string)	When using multiple currencies, some reports allow you to display converted amounts by selecting the appropriate column to display. For example, in opportunity reports, you can include the Amount (converted) column on the report. This field is an enumeration of type string that defines the currency in which to display converted amounts. Valid values: Must be one of the valid alphabetic, three-letter currency ISO codes defined by the ISO 4217 standard, such as `USD`, `GBP`, or `JPY`.
`description`	string	Specifies a general description, which is displayed with the report name. Maximum characters: 255 characters.
`division`	string	If your organization uses divisions to segment data and you have the "Affected by Divisions" permission, records in the report must match this division. This field is available in API version 17.0 and later.

`filter`	ReportFilter	Limits report results to records with specific data. For example, you can limit report results to opportunities for which the amount is greater than $1,000:
		`<filter>`
		`  <criteriaItems>`
		`    <column>AMOUNT</column>`
		
		`<operator>greaterThan</operator>`
		
		`    <value>1000</value>`
		`  </criteriaItems>`
		`</filter>`
		
		For more information, see "Entering Filter Criteria" in the Salesforce.com online help.

`format`	ReportFormat (enumeration of type string)	Defines the report format. For example, `Tabular` for a simple data list without subtotals.

Field	Field Type	Description
`fullName`	string	The report unique developer name used as an identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`groupingsAcross`	ReportGrouping[]	List that defines the fields by which you want to group and subtotal data across a matrix report (row headings). When grouping by a date field, you can further group the data by a specific time period such as days, weeks, or months. Maximum: 2 fields.
`groupingsDown`	ReportGrouping[]	For Summary and Matrix reports: List that defines the fields by which you want to group and subtotal. For summary reports, choosing more than one sort field allows you to subsort your data. For matrix reports, specifies summary fields for column headings. When grouping by a date field, you can further group the data by a specific time period such as days, weeks, or months. Maximum for matrix reports: 2. Maximum for summary reports: 3
`name`	string	Required. The report name. for example, `Opportunity Pipeline`
`params`	ReportParam[]	List that specifies settings specific to each report type, in particular options that let you filter a report to obtain useful subsets. For example, the Activities report type lets you specify whether you want to see open or closed activities or both and whether you want to see tasks or events or both. Valid values depend on the report type.
`reportType`	string	Required. Defines the type of data in the report. For example, `Opportunity` to create a report of opportunities data.
`roleHierarchyFilter`	string	The role name for a report drill down. Some reports, such as opportunity and activity reports, display Hierarchy links that allow you to drill down to different data sets based on the role hierarchy. This field is available in API version 17.0 and later.
`rowLimit`	int	Defines the maximum number of rows that can be returned for the report.

Field	Field Type	Description
`scope`	string	Defines the scope of data on which you run the report. For example, whether you want to run the report against all opportunities, opportunities you own, or opportunities your team owns. Valid values depend on the `reportType`. For example, for Accounts reports: • `MyAccounts` • `MyTeamsAccounts` • `AllAccounts`
`showDetails`	boolean	`false` shows a collapsed view of the report with only the headings, subtotals, and total. Default: `true`
`sortColumn`	string	Specifies the field on which to sort data in the report. Use `sortOrder` to specify the sort order.
`sortOrder`	SortOrder (enumeration of type string)	Specifies the sort order. Use `sortColumn` to specify the field on which to sort.
`territoryHierarchyFilter`	string	The territory name for a report drill down. If your organization uses territory management, some reports display Hierarchy links that allow you to drill down to different data sets based on the territory hierarchy. This field is available in API version 17.0 and later.
`timeFrameFilter`	ReportTimeFrameFilter	Limits report results to records within a specified time frame.
`userFilter`	string	The user name for a report drill down. Some reports, such as opportunity and activity reports, display Hierarchy links that allow you to drill down to different data sets based on the user hierarchy. This field is available in API version 17.0 and later.

ReportAggregate

ReportAggregate defines custom summary formulas on summary or matrix reports. For more information on these fields, see "Building Custom Summary Formulas" in the Salesforce.com online help.

Field	Field Type	Description
`acrossGroupingContext`	string	Defines the row grouping level at which you want your custom summary formula to be displayed. This is a new field in API version 15.0.

Field	Field Type	Description
`calculatedFormula`	string	Required. The custom summary formula. For example, `AMOUNT:SUM + OPP_QUANTITY:SUM`
`datatype`	ReportAggregateDatatype (enumeration of type string)	Required. Specifies the data type for formatting and display of the custom summary formula results.
`description`	string	The custom summary formula description. Maximum: 255 characters.
`developerName`	string	Required. The internal development name of the custom summary formula, for example, `FORMULA1`. This is used to reference custom summary formulas from other report components, including conditional highlighting.
`downGroupingContext`	string	Defines the column grouping level at which you want your custom summary formula to be displayed. This is a new field in API version 15.0.
`isActive`	boolean	Required. `true` displays the formula result in the report. `false` does not display the result in the report.
`masterLabel`	string	Required. The custom summary formula label (name).
`scale`	int	The formula result is calculated to the specified number of decimal places. Valid values `0` through `18`.

ReportGrouping

ReportGrouping defines how to group and subtotal data for summary and matrix reports.

Field	Field Type	Description
`dateGranularity`	UserDateGranularity (enumeration of type string)	When grouping by a date field, the time period by which to group.
`field`	string	Required. The field by which you want to summarize data. For example, `CAMPAIGN_SOURCE`
`sortOrder`	SortOrder	Required. Whether to sort data in ascending or descending alphabetical and numerical order.

SortOrder

An enumeration of type string that defines the order in which data is sorted in the report fields. Valid values:

Field	Description
`Asc`	Sorts data in ascending alphabetical and numerical order.
`Desc`	Sorts data in descending alphabetical and numerical order.

UserDateGranularity

An enumeration of type string that defines the time period by which to group data. Valid values:

Enumeration Value	Description
`None`	No grouping by date
`Day`	By day
`Week`	By week
`Month`	By month
`Quarter`	By quarter
`Year`	By year
`FiscalQuarter`	By fiscal quarter. You can set the fiscal year for your organization. See "Setting the Fiscal Year" in the Salesforce.com online help.
`FiscalYear`	By fiscal year
`MonthInYear`	By calendar month in year
`DayInMonth`	By calendar day in month
`FiscalPeriod`	When custom fiscal years are enabled: By fiscal period
`FiscalWeek`	When custom fiscal years are enabled: By fiscal week

ReportSummaryType

An enumeration of type string that defines how report fields are summarized. Valid values:

Enumeration Value	Description
`Sum`	Total
`Average`	Average
`Maximum`	Largest value
`Minimum`	Smallest value
`None`	The field is not summarized.

ReportColorRange

ReportColorRange defines conditional highlighting for report summary data.

Field	Field Type	Description
`aggregate`	ReportSummaryType (enumeration of type string)	Required. Defines how the field specified in `columnName` is summarized. For example, `Sum`.
`columnName`	string	Required. Specifies the field whose value ranges are represented by colors.
`highBreakpoint`	double	Required. Specifies the number that separates the mid color from the high color.
`highColor`	string	Required. Specifies the color (in HTML format) to represent data that falls into the high number range. This color spans from the `highBreakpoint` value.

Field	Field Type	Description
`lowBreakpoint`	double	Required. Specifies the number that separates the low color from the mid color.
`lowColor`	string	Required. Specifies a color (in HTML format) to represent data that falls into the low value range, below the `lowBreakpoint` value.
`midColor`	string	Required. Specifies a color (in HTML format) to represent data that falls into the mid value range.

ReportColumn

ReportColumn defines how fields (columns) are displayed in the report.

Field	Field Type	Description
`aggregateTypes`	ReportSummaryType[] (enumeration of type string)	List that defines if and how each report field is summarized.
`field`	string	Required. The field name. For example, `AGE` or `OPPORTUNITY_NAME`

ReportFilter

ReportFilter limits the report results by filtering data on specified fields.

Field	Field Type	Description
`booleanFilter`	string	Specifies advanced filter conditions. For more information on advanced filter conditions, see "Working with Advanced Filter Conditions" in the Salesforce.com online help.
`criteriaItems`	ReportFilterItem	The criteria by which you want to filter report data.
`language`	Language (enumeration of type string)	The language used when a report filters against a picklist value using the operators `contains` or `startsWith`. For a list of valid language values, see Language.

ReportFilterItem

ReportFilterItem limits the report results by filtering data on specified fields.

Field	Field Type	Description
`column`	string	Required. The field on which you want to filter data. For example, `AMOUNT`
`operator`	FilterOperation (enumeration of type string)	Required. An enumeration of type string that defines the operator used to filter the data, for example, `greaterThan`. For valid values, see FilterOperation.
`value`	string	The value by which you want to filter the data, for example, `1000`. Note that the Metadata API filter condition values do not always match those that you enter in the report

Field	Field Type	Description
		wizard. For example, in the Metadata API dates are always converted to the US date format and values entered in a non-US English language may be converted to a standard US English equivalent.

ReportFormat

An enumeration of type string that defines the report format. Valid values:

Enumeration Value	Description
`Matrix`	Summarizes data in a grid. Use to compare related totals.
`Summary`	Lists, sorts, and subtotals data.
`Tabular`	Lists data with no sorting or subtotals.

ReportParam

ReportParam represents settings specific to a report type, especially options that let you filter a report to certain useful subsets.

Field	Field Type	Description
`name`	string	Required. Specifies a specific `reportType` setting.
`value`	string	Required. The setting value.

ReportAggregateDatatype

An enumeration of type string that specifies the data type for formatting and display of custom summary formula results. Valid values:

Enumeration Value
`currency`
`number`
`percent`

ReportChart

ReportChart represents charts on summary and matrix reports.

Field	Field Type	Description
`backgroundColor1`	string	Specifies the beginning color (in HTML format) for a gradient color background.
`backgroundColor2`	string	Specifies the end color (in HTML format) for a gradient color background.
`backgroundFadeDir`	ChartBackgroundDirection (enumeration of type string)	Specifies the direction for a gradient color background. Use with `backgroundColor1` to specify the beginning color

Field	Field Type	Description
		and `backgroundColor2` to specify the end color for the gradient design. Use white for both if you do not want a background design. Valid values: • `diagonal` • `leftToRight` • `topToBottom`
`chartSummaries`	ChartSummary[]	Specifies the summaries you want to use for the chart. Invalid summaries are ignored without notification. If there are no valid summaries, RowCount is used by default for the axis value. This field is available in API version 17.0 and later.
`chartType`	ChartType (enumeration of type string)	Required. Specifies the chart type. Available chart types depend on the report type.
`enableHoverLabels`	boolean	Specifies whether to display values, labels, and percentages when hovering over charts. Hover details depend on chart type. Percentages apply to pie, donut, and funnel charts only. This field is available in API version 17.0 and later.
`expandOthers`	boolean	Specifies whether to combine all groups less than or equal to 3% of the total into a single 'Others' wedge or segment. This only applies to pie, donut, and funnel charts. Set to `true` to show all values individually on the chart; set to `false` to combine small groups into 'Others.' This field is available in API version 17.0 and later.
`groupingColumn`	string	Specifies the field by which to group data. This data is displayed on the X-axis for vertical column charts and on the Y-axis for horizontal bar charts.
`legendPosition`	ChartLegendPosition (enumeration of type string)	Required. The location of the legend with respect to the chart. The valid values are: • `Bottom` • `OnChart` • `Right`
`location`	ChartPosition (enumeration of type string)	Required. Specifies whether the chart is displayed at the top or bottom of the report.
`secondaryGroupingColumn`	string	For grouped chart types: Specifies the field by which to group the data.
`showAxisLabels`	boolean	For bar and line charts: Specifies whether the chart displays names for each axis.
`showPercentage`	boolean	Indicates if percentages are displayed for wedges and segments of pie, donut, and funnel charts, as well as for gauges (`true`), or not (`false`).

Field	Field Type	Description
`showTotal`	boolean	Indicates if the total is displayed for donut charts and gauges (`true`), or not (`false`).
`showValues`	boolean	Indicates if the values of individual records or groups are displayed for charts (`true`), or not (`false`).
`size`	ReportChartSize (enumeration of type string)	Required. Specifies the chart size.
`summaryAggregate`	ReportSummaryType (enumeration of type string)	Defines how to summarize the chart data. For example, `Sum`. No longer supported in version API 17.0 and later. See `chartSummaries`.
`summaryAxisManualRangeEnd`	double	When specifying the axis range manually: Defines the ending value.
`summaryAxisManualRangeStart`	double	When specifying the axis range manually: Defines the starting value.
`summaryAxisRange`	ChartRangeType (enumeration of type string)	Required. For bar, line, and column charts: Defines whether to specify the axis range manually or automatically.
`summaryColumn`	string	Required. Specifies the field by which to summarize the chart data. Typically this field is displayed on the Y-axis. No longer supported in version API 17.0 and later. See `chartSummaries`.
`textColor`	string	The color (in HTML format) of the chart text and labels.
`textSize`	int	The size of the chart text and labels. Valid values: • `8` • `9` • `10` • `12` • `14` • `18` • `24` • `36` The maximum size is 18. Larger values are shown at 18 points.
`title`	string	The chart title. Max 255 characters.
`titleColor`	string	The color (in HTML format) of the title text.
`titleSize`	int	The size of the title text. Valid values: • `8` • `9` • `10` • `12` • `14` • `18` • `24` • `36`

Field	Field Type	Description
		The maximum size is 18. Larger values are shown at 18 points.

ChartType

An enumeration of type string that defines the chart type. For information on each of these chart types, see “Chart Types” in the Salesforce.com online help. Valid values:

Enumeration Value
`None`
`HorizontalBar`
`HorizontalBarGrouped`
`HorizontalBarStacked`
`HorizontalBarStackedTo100`
`VerticalColumn`
`VerticalColumnGrouped`
`VerticalColumnStacked`
`VerticalColumnStackedTo100`
`Line`
`LineGrouped`
`LineCumulative`
`LineCumulativeGrouped`
`Pie`
`Donut`
`Funnel`
`VerticalColumnLine`
`VerticalColumnGroupedLine`
`VerticalColumnStackedLine`

ChartPosition

An enumeration of type string that specifies the position of the chart in the report. Valid values:

Enumeration Value
`CHART_TOP`
`CHART_BOTTOM`

ChartSummary

ChartSummary defines how data in the chart is summarized. Valid values:

Field	Field Type	Description
aggregate	ReportSummaryType	Specifies the aggregation method—such as Sum, Average, Min, and Max—for the summary value. Use the column field to specify the summary value to use for the aggregation. You don't need to specify this field for RowCount or custom summary formulas.
axisBinding	ChartAxis	Specifies the axis or axes to use on the chart. Use the column field to specify the summary value to use for the axis.
column	string	Required. Specifies the summary field for the chart data. If all columns are invalid, RowCount is used by default for the axis value. For vertical column and horizontal bar combination charts, you can specify up to four values.

ChartAxis

An enumeration of type string that specifies the axis or axes to be used in charts. Valid values:

Enumeration Value	Description
y	The Y-axis for the chart.
y2	The secondary Y-axis for vertical column combination charts with a line added.

ReportChartSize

An enumeration of type string that specifies the chart size. Valid values:

Enumeration Value
Tiny
Small
Medium
Large
Huge

ChartRangeType

An enumeration of type string that defines the report format. Valid values:

Enumeration Value
Manual
Automatic

ReportTimeFrameFilter

ReportTimeFrameFilter represents the report time period.

Field	Field Type	Description
`dateColumn`	string	Required. The date field on which to filter data. For example, `CLOSE_DATE`
`endDate`	date	When `interval` is `INTERVAL_CUSTOM`, specifies the end of the custom time period.
`interval`	UserDateInterval (enumeration of type string)	Required. Specifies the period of time.
`startDate`	date	When `interval` is `INTERVAL_CUSTOM`, specifies the start of the custom time period.

UserDateInterval

An enumeration of type string that defines the period of time. Valid values:

Enumeration Value	Description
`INTERVAL_CURRENT`	Current fiscal quarter
`INTERVAL_CURNEXT1`	Current and next fiscal quarters
`INTERVAL_CURPREV1`	Current and previous fiscal quarters
`INTERVAL_NEXT1`	Next fiscal quarter
`INTERVAL_PREV1`	Previous fiscal quarter
`INTERVAL_CURNEXT3`	Current and next three fiscal quarters
`INTERVAL_CURFY`	Current fiscal year
`INTERVAL_PREVFY`	Previous fiscal year
`INTERVAL_PREV2FY`	Previous two fiscal years
`INTERVAL_AGO2FY`	Two fiscal years ago
`INTERVAL_NEXTFY`	Next fiscal year
`INTERVAL_PREVCURFY`	Current and previous fiscal years
`INTERVAL_PREVCUR2FY`	Current and previous two fiscal years
`INTERVAL_CURNEXTFY`	Current and next fiscal year
`INTERVAL_CUSTOM`	A custom time period. Use `startDate` and `endDate` fields to specify the time period's start date and end date.
`INTERVAL_YESTERDAY`	Yesterday
`INTERVAL_TODAY`	Today
`INTERVAL_TOMORROW`	Tomorrow
`INTERVAL_LASTWEEK`	Last calendar week

Enumeration Value	Description
INTERVAL_THISWEEK	This calendar week
INTERVAL_NEXTWEEK	Next calendar week
INTERVAL_LASTMONTH	Last calendar month
INTERVAL_THISMONTH	This calendar month
INTERVAL_NEXTMONTH	Next calendar month
INTERVAL_LASTTHISMONTH	Current and previous calendar months
INTERVAL_THISNEXTMONTH	Current and next calendar months
INTERVAL_CURRENTQ	Current calendar quarter
INTERVAL_CURNEXTQ	Current and next calendar quarters
INTERVAL_CURPREVQ	Current and previous calendar quarters
INTERVAL_NEXTQ	Next calendar quarter
INTERVAL_PREVQ	Previous calendar quarter
INTERVAL_CURNEXT3Q	Current and next three calendar quarters
INTERVAL_CURY	Current calendar year
INTERVAL_PREVY	Previous calendar year
INTERVAL_PREV2Y	Previous two calendar years
INTERVAL_AGO2Y	Two calendar years ago
INTERVAL_NEXTY	Next calendar year
INTERVAL_PREVCURY	Current and previous calendar years
INTERVAL_PREVCUR2Y	Current and previous two calendar years
INTERVAL_CURNEXTY	Current and next calendar years
INTERVAL_LAST7	Last 7 days
INTERVAL_LAST30	Last 30 days
INTERVAL_LAST60	Last 60 days
INTERVAL_LAST90	Last 90 days
INTERVAL_LAST120	Last 120 days
INTERVAL_NEXT7	Next 7 days
INTERVAL_NEXT30	Next 30 days
INTERVAL_NEXT60	Next 60 days
INTERVAL_NEXT90	Next 90 days
INTERVAL_NEXT120	Next 120 days
LAST_FISCALWEEK	When custom fiscal years are enabled: Last fiscal week
THIS_FISCALWEEK	When custom fiscal years are enabled: This fiscal week

Enumeration Value	Description
NEXT_FISCALWEEK	When custom fiscal years are enabled: Next fiscal week
LAST_FISCALPERIOD	When custom fiscal years are enabled: Last fiscal period
THIS_FISCALPERIOD	When custom fiscal years are enabled: This fiscal period
NEXT_FISCALPERIOD	When custom fiscal years are enabled: Next fiscal period
LASTTHIS_FISCALPERIOD	When custom fiscal years are enabled: This fiscal period and last fiscal period
THISNEXT_FISCALPERIOD	When custom fiscal years are enabled: This fiscal period and next fiscal period
CURRENT_ENTITLEMENT_PERIOD	Current entitlement period
PREVIOUS_ENTITLEMENT_PERIOD	Previous entitlement period
PREVIOUS_TWO_ENTITLEMENT_PERIODS	Previous two entitlement periods
TWO_ENTITLEMENT_PERIODS_AGO	Two entitlement periods ago
CURRENT_AND_PREVIOUS_ENTITLEMENT_PERIOD	Current and previous entitlement period
CURRENT_AND_PREVIOUS_TWO_ENTITLEMENT_PERIODS	Current and previous two entitlement periods

Declarative Metadata Sample Definition

A sample XML report definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<Report xmlns="http://soap.sforce.com/2006/04/metadata">
    <aggregates>
        <acrossGroupingContext>CRT_Object__c$Id</acrossGroupingContext>
        <calculatedFormula>PREVGROUPVAL(CRT_Object__c.Currency__c:AVG, CRT_Object__c.Id) *
                PARENTGROUPVAL(CRT_Object__c.Number__c:MAX, CRT_Object__c.CreatedBy.Name,
                COLUMN_GRAND_SUMMARY)/RowCount</calculatedFormula>
        <datatype>number</datatype>
        <developerName>FORMULA1</developerName>
        <downGroupingContext>CRT_Object__c$CreatedBy</downGroupingContext>
        <isActive>true</isActive>
        <masterLabel>CurrCSF</masterLabel>
        <scale>2</scale>
    </aggregates>
    <aggregates>
        <acrossGroupingContext>CRT_Object__c$LastModifiedDate</acrossGroupingContext>
        <calculatedFormula>IF(RowCount&gt;10,
                BLANKVALUE(ROUND(PREVGROUPVAL(CRT_Object__c.Currency__c:SUM,
                CRT_Object__c.LastModifiedDate),3),
                PARENTGROUPVAL(CRT_Object__c.Number__c:SUM, ROW_GRAND_SUMMARY,
                CRT_Object__c.Id))  , 1000)</calculatedFormula>
        <datatype>number</datatype>
        <developerName>FORMULA2</developerName>
        <downGroupingContext>GRAND_SUMMARY</downGroupingContext>
        <isActive>true</isActive>
        <masterLabel>numCSF</masterLabel>
        <scale>2</scale>
    </aggregates>
    <chart>
        <backgroundColor1>#FFFFFF</backgroundColor1>
        <backgroundColor2>#FFFFFF</backgroundColor2>
        <backgroundFadeDir>Diagonal</backgroundFadeDir>
        <chartSummaries>
```

```
            <axisBinding>y</axisBinding>
            <column>FORMULA1</column>
        </chartSummaries>
        <chartSummaries>
            <axisBinding>y</axisBinding>
            <column>FORMULA2</column>
        </chartSummaries>
        <chartSummaries>
            <aggregate>Maximum</aggregate>
            <axisBinding>y</axisBinding>
            <column>CRT_Object__c$Number__c</column>
        </chartSummaries>
        <chartSummaries>
            <axisBinding>y</axisBinding>
            <column>RowCount</column>
        </chartSummaries>
        <chartType>VerticalColumn</chartType>
        <groupingColumn>CRT_Object__c$LastModifiedDate</groupingColumn>
        <legendPosition>Right</legendPosition>
        <location>CHART_TOP</location>
        <size>Medium</size>
        <summaryAxisRange>Auto</summaryAxisRange>
        <textColor>#000000</textColor>
        <textSize>12</textSize>
        <titleColor>#000000</titleColor>
        <titleSize>18</titleSize>
    </chart>
    <columns>
        <field>CRT_Object__c$Name</field>
    </columns>
    <columns>
        <aggregateTypes>Average</aggregateTypes>
        <field>CRT_Object__c$Currency__c</field>
    </columns>
    <columns>
        <aggregateTypes>Maximum</aggregateTypes>
        <field>CRT_Object__c$Number__c</field>
    </columns>
    <format>Matrix</format>
    <groupingsAcross>
        <dateGranularity>Day</dateGranularity>
        <field>CRT_Object__c$Id</field>
        <sortOrder>Asc</sortOrder>
    </groupingsAcross>
    <groupingsAcross>
        <dateGranularity>Year</dateGranularity>
        <field>CRT_Object__c$LastModifiedDate</field>
        <sortOrder>Asc</sortOrder>
    </groupingsAcross>
    <groupingsDown>
        <dateGranularity>Day</dateGranularity>
        <field>CRT_Object__c$CreatedBy</field>
        <sortOrder>Asc</sortOrder>
    </groupingsDown>
    <groupingsDown>
        <dateGranularity>Day</dateGranularity>
        <field>CRT_Object__c$Currency__c</field>
        <sortOrder>Desc</sortOrder>
    </groupingsDown>
    <name>CrtMMVC</name>
    <reportType>CRT1__c</reportType>
    <scope>organization</scope>
    <showDetails>false</showDetails>
    <timeFrameFilter>
        <dateColumn>CRT_Object__c$CreatedDate</dateColumn>
        <interval>INTERVAL_CUSTOM</interval>
    </timeFrameFilter>
</Report>
```

See Also:

Dashboard

ReportType

Represents the metadata associated with a custom report type. It extends the Metadata metadata type and inherits its `fullName` field. Custom report types allow you to build a framework from which users can create and customize reports. For more information, see "Custom Report Types Overview" in the Salesforce.com online help.

Declarative Metadata File Suffix and Directory Location

The file suffix is `.reportType` for the custom report type definition. There is one file per custom report type. Report types are stored in the `reportTypes` directory of the corresponding package directory.

Version

Custom report types are available in API version 14.0 and later.

Fields

Field Name	Field Type	Description
`baseObject`	string	Required. The primary object for the custom report type, for example, Account. All objects, including custom objects, are supported. You cannot edit this field after initial creation.
`category`	ReportTypeCategory (enumeration of type string)	Required. This field controls the category for the report in the report wizard. The valid values are: • `accounts` • `opportunities` • `forecasts` • `cases` • `leads` • `campaigns` • `activities` • `busop` • `products` • `admin` • `territory` • `other` • `content`
`deployed`	boolean	Required. Indicates whether the report type is available for use in the report wizard (`true`) or whether it's still in development (`false`).

Field Name	Field Type	Description
`description`	string	The description of the custom report type.
`fullName`	string	The report type developer name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
`join`	ObjectRelationship	The object joined to the `baseObject`. For example, Contacts may be joined to the primary Accounts object.
`label`	string	Required. The report type label.
`sections`	ReportLayoutSection[]	The groups of columns available for the report type. Though columns are not strictly required, a report without columns is not very useful.

ObjectRelationship

ObjectRelationship represents a join to another object. For more information, see "Choosing Object Relationships for Custom Report Types" in the Salesforce.com online help.

Field Name	Field Type	Description
`join`	ObjectRelationship	This field is a recursive reference that allows you to join more than two objects. A maximum of four objects can be joined in a custom report type. When more than two objects are joined, an inner join is not allowed if there has been an outer join earlier in the join sequence. The `baseObject` is first joined to the object specified in `relationship`; the resulting data set is then joined with any objects specified in this field.
`outerJoin`	boolean	Required. Indicates whether this is an outer join (`true`) or not (`false`). An outer join returns a row even if the joined table does not contain a matching value in the join column.
`relationship`	string	Required. The object joined to the primary object; for example, Contacts.

ReportLayoutSection

ReportLayoutSection represents a group of columns used in the custom report type.

Field Name	Field Type	Description
`columns`	ReportTypeColumn[]	The list of columns projected from the query, defined by this custom report type.
`masterLabel`	string	Required. The label for this group of columns in the report wizard.

ReportTypeColumn

ReportTypeColumn represents a column in the custom report type.

Field Name	Field Type	Description
checkedByDefault	boolean	Required. Indicates whether this column is selected be default (`true`) or not (`false`).
displayNameOverride	string	A customized column name, if desired.
field	string	Required. The field name associated with the report column.
table	string	Required. The table associated with the field; for example, Account.

Declarative Metadata Sample Definition

The definition of a custom report type is shown below. Account is joined to Contacts and the resulting data set is joined with Assets.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReportType xmlns="http://soap.sforce.com/2006/04/metadata">
    <baseObject>Account</baseObject>
    <category>accounts</category>
    <deployed>true</deployed>
    <description>Account linked to Contacts and Assets</description>
    <join>
        <join>
            <outerJoin>false</outerJoin>
            <relationship>Assets</relationship>
        </join>
        <outerJoin>false</outerJoin>
        <relationship>Contacts</relationship>
    </join>
    <label>Account Contacts and Assets</label>
    <sections>
        <columns>
            <checkedByDefault>true</checkedByDefault>
            <field>obj_lookup__c.Id</field>
            <table>Account</table>
        </columns>
        <columns>
            <checkedByDefault>false</checkedByDefault>
            <field>obj_lookup__c.Name</field>
            <table>Account</table>
        </columns>
        <columns>
            <checkedByDefault>false</checkedByDefault>
            <field>Opportunity__c.Amount</field>
            <table>Account</table>
        </columns>
        <columns>
            <checkedByDefault>false</checkedByDefault>
            <field>Owner.IsActive</field>
            <table>Account</table>
        </columns>
        <masterLabel>Accounts</masterLabel>
    </sections>
    <sections>
        <columns>
            <checkedByDefault>false</checkedByDefault>
            <field>Owner.Email</field>
            <table>Account.Contacts</table>
        </columns>
        <columns>
            <checkedByDefault>false</checkedByDefault>
            <field>byr__c</field>
            <table>Account.Contacts</table>
        </columns>
        <columns>
            <checkedByDefault>true</checkedByDefault>
            <field>ReportsTo.CreatedBy.Contact.Owner.MobilePhone</field>
            <table>Account.Contacts</table>
        </columns>
        <masterLabel>Contacts</masterLabel>
    </sections>
</ReportType>
```

Scontrol

Important: 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.

Represents an Scontrol component, corresponding to an s-control in the Salesforce.com user interface. For more information, see “About S-Controls” in the Salesforce.com online help. This metadata type extends the MetadataWithContent component and shares its fields.

Declarative Metadata File Suffix and Directory Location

The file suffix is `.scf` for the s-control file. The accompanying metadata file is named *`ScontrolName`*`-meta.xml`.

Scontrol components are stored in the `scontrols` folder in the corresponding package directory.

Version

Scontrols are available in API version 10.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`content`	base64Binary	Content of the s-control. Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion is usually handled for you by a SOAP client. This field is inherited from the MetadataWithContent component.
`contentSource`	SControlContentSource (enumeration of type string)	Required. Determines how you plan to use the s-control: • `HTML`: Select this option if you want to enter the content for your s-control in `content`. • `URL`: Select this option if you want to enter the link or URL of an external website in `content`. • `Snippet`: Snippets are s-controls that are designed to be included in other s-controls. Select this option if you want to enter the content for your s-control snippet in `content`.
`description`	string	Optional text that describes the s-control. This only displays to users with “View All Data” permission (administrator).

Field Name	Field Type	Description
encodingKey	Encoding (enumeration of type string)	Required. The default encoding setting is Unicode: UTF-8. Change it if you are passing information to a URL that requires data in a different format. This option is available when you select URL as the value for contentSource.
fileContent	base64	File contents displayed if you add this s-control to a custom link. The file can contain a Java applet, Active-X control, or any other type of content you want. This option only applies to s-controls with a value of HTML for contentSource.
fileName	string	The unique name for the s-control. This name can contain only underscores and alphanumeric characters, and must be unique in your organization. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field cannot be changed for components installed by a managed package. It is only relevant if the fileContent field also has a value. This is a new field in API version 14.0.
fullName	string	The s-control developer name used as a unique identifier for API access. The fullName can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. If this field contained characters before version 14.0 that are no longer allowed, the characters were stripped out of this field, and the previous value of the field was saved in the name field. This field is inherited from the Metadata component.
name	string	Required. The unique name for the s-control. It must contain alphanumeric characters only and begin with a letter. For example example_s_control.
supportsCaching	boolean	Required. Indicates whether the s-control supports caching (true) or not (false). Caching optimizes the page so that it remembers which s-controls are on the page when it reloads. This option only applies to HTML s-controls.

Declarative Metadata Sample Definition

The following sample creates the Myriad_Publishing.scf s-control, which creates a link to the website specified in the s-control. The corresponding Myriad_Publishing.scf-meta.xml metadata file follows the s-control file.

Myriad_Publishing.scf file:

```
http://www.myriadpubs.com
```

Myriad_Publishing.scf-meta.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<Scontrol xmlns="http://soap.sforce.com/2006/04/metadata">
    <contentSource>URL</contentSource>
    <description>s-control to open Myriad Publishing website.</description>
    <encodingKey>UTF-8</encodingKey>
    <name>Myriad Publishing</name>
    <supportsCaching>true</supportsCaching>
</Scontrol>
```

StaticResource

Represents a static resource file, often a code library in a ZIP file. This metadata type extends the MetadataWithContent component and shares its fields.

Static resources allow you to upload content that you can reference in a Visualforce page, including archives (such as .zip and .jar files), images, stylesheets, JavaScript, and other files.

File Suffix and Directory Location

The file suffix is `.resource` for the template file. The accompanying metadata file is named *`resource`*`-meta.xml`.

Static resource components are stored in the `staticresources` folder in the corresponding package directory.

Version

Static resources are available in API version 12.0 and later.

Fields

This metadata type contains the following fields:

Field Name	Field Type	Description
`cacheControl`	StaticResourceCacheControl (enumeration of type string)	Required. Indicates whether the static resource is marked with a public caching tag so that a third-party delivery client can cache the content. This is a new field in API version 14.0. The valid values are: • Private • Public
`content`	base64Binary	The static resource content. Base 64-encoded binary data. Prior to making an API call, client applications must encode the binary attachment data as base64. Upon receiving a response, client applications must decode the base64 data to binary. This conversion is usually handled for you by a SOAP client. This field is inherited from the MetadataWithContent component.
`contentType`	string	Required. The content type of the file, for example text/plain.
`description`	string	The description of the static resource.
`fullName`	string	The static resource name. The name can only contain characters, letters, and the underscore (_) character, must start with a letter, and cannot end with an underscore or contain two consecutive underscore characters. Inherited from the Metadata component, this field is not defined in the WSDL for this component. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.

Declarative Metadata Sample Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<StaticResource xmlns="http://soap.sforce.com/2006/04/metadata">
    <contentType>text/plain</contentType>
    <description>Test Resource</description>
</StaticResource>
```

Translations

This metadata type allows you to work with translations for a variety of languages. The supported languages are listed in Language on page 205. It extends the Metadata metadata type and inherits its `fullName` field. The ability to translate component labels is part of the Translation Workbench. For more information, see “Setting Up the Translation Workbench” in the Salesforce.com online help.

Language

Languages are identified by a five-character *locale* code or a two-character language code. A complete locale includes a two-character ISO language code, followed by an underscore (_) and a two-character ISO country code. For example, the locale is “en_US” for the United States. Some of the locales below only specify a two-character ISO language code, for example “fr”.

Salesforce.com is available in 16 fully supported languages.

- Danish: `da`
- German: `de`
- English: `en_US`
- Spanish: `es`
- Finnish: `fi`
- French: `fr`
- Italian: `it`
- Japanese: `ja`
- Korean: `ko`
- Dutch: `nl_NL`
- Portuguese (Brazil): `pt_BR`
- Russian: `ru`
- Swedish: `sv`
- Thai: `th`
- Chinese (Simplified): `zh_CN`
- Chinese (Traditional): `zh_TW`

Salesforce.com supports end-user languages, where administration pages and online help are not translated.

- Arabic: `ar`
- Bulgarian: `bg`
- Czech: `cs`
- English (UK): `en_GB`
- Greek: `el`
- Spanish (Mexico): `es_MX`
- Hebrew: `iw`
- Hungarian: `hu`

- Indonesian: `in`
- Polish: `pl`
- Romanian: `ro`
- Turkish: `tr`
- Ukrainian: `uk`
- Vietnamese: `vi`

For organizations that use Salesforce.com exclusively as a platform, Salesforce.com provides platform languages. Platform languages are languages that you can translate your customizations into but Salesforce.com provides no default translations. Administrators must contact their salesforce.com representative to enable platform languages.

Platform languages are available in all of the places one can select language in the application, but selecting a platform language will default all labels in the application to English. All customizations made to Salesforce.com can be translated into a platform language and renaming can be used to provide translations for standard field names on most objects. However, informative text and non-field label text is not translatable.

- English (Australia): `en_AU`
- English (Canada): `en_CA`
- English (India): `en_IN`
- English (Malaysia): `en_MY`
- English (Philippines): `en_PH`
- French (Canada): `fr_CA`
- Georgian: `ka`
- Norwegian: `no`
- Serbian (Latin): `sh`
- Slovak: `sk`
- Serbian (Cyrillic): `sr`

Declarative Metadata File Suffix and Directory Location

Translations are stored in a file with a format of `localeCode.translation`, where `localeCode` is the locale code of the translation language. For example, the file name for German translations is `de.translation`. The supported locale codes are listed in Language on page 205.

Custom object translations are stored in the `translations` folder in the corresponding package directory.

Version

Translations components are available in API version 14.0 and later.

Fields

Field	Field Type	Description
`customApplications`	CustomApplicationTranslation[]	A list of custom application translations.
`customLabels`	CustomLabelTranslation[]	A list of custom label translations.
`customPageWebLinks`	CustomPageWebLinkTranslation[]	A list of translations for web links defined in a home page component.
`customTabs`	CustomTabTranslation[]	A list of custom tab translations.

Field	Field Type	Description
`fullName`	string	Required. The language code; for example, `de` for German. Inherited from Metadata, this field is not defined in the WSDL for this metadata type. It must be specified when creating, updating, or deleting. See `create()` to see an example of this field specified for a call.
`reportTypes`	ReportTypeTranslation[]	A list of report type translations.
`scontrols`	ScontrolTranslation[]	A list of s-control translations.

CustomApplicationTranslation

CustomApplicationTranslation contains details for a custom application translation. For more details, see CustomApplication on page 87.

Field	Field Type	Description
`label`	string	Required. The translated custom application name. Maximum of 765 characters.
`name`	string	Required. The name of the custom application.

CustomLabelTranslation

CustomLabelTranslation contains details for a custom label translation. For more details, see CustomLabels on page 88.

Field	Field Type	Description
`label`	string	Required. The translated custom label name. Maximum of 765 characters.
`name`	string	Required. The custom label name.

CustomPageWebLinkTranslation

CustomPageWebLinkTranslation contains details for a translation of a web link defined in a home page component. For more details, see CustomPageWebLink on page 131.

Field	Field Type	Description
`label`	string	Required. The translated web link.
`name`	string	Required. The name of the web link.

CustomTabTranslation

CustomTabTranslation contains details for a translation of a custom tab. For more details, see CustomTab on page 137.

Field	Field Type	Description
`label`	string	Required. The translated custom tab name.
`name`	string	Required. The custom tab name.

ReportTypeTranslation

ReportTypeTranslation contains details for a translation of a custom report type. For more details, see ReportType on page 199.

Field	Field Type	Description
`description`	string	The translated report type description.
`label`	string	The translated report type name.
`name`	string	Required. The name of the report type.
`sections`	ReportTypeSectionTranslation[]	A list of report type section translations.

ReportTypeSectionTranslation

ReportTypeSectionTranslation contains details for a report type section translation.

Field	Field Type	Description
`columns`	ReportTypeColumnTranslation[]	A list of report type column translations.
`label`	string	The translated report type section name.
`name`	string	Required. The name of the report type section.

ReportTypeColumnTranslation

ReportTypeColumnTranslation contains details for a report type column translation.

Field	Field Type	Description
`label`	string	Required. The translated report type column name.
`name`	string	Required. The report type column name.

ScontrolTranslation

Important: 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.

ScontrolTranslation contains details for a translation of an s-control. For more information, see “About S-Controls” in the Salesforce.com online help.

Field	Field Type	Description
`label`	string	Required. The translated s-control name.

Field	Field Type	Description
name	string	Required. The name of the s-control.

Declarative Metadata Sample Definition

A sample XML definition of a translations component is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Translations xmlns="http://soap.sforce.com/2006/04/metadata">
    <customApplications>
    <label>Angebot-Manager</label>
        <name>Quote Manager</name>
    </customApplications>
    <customLabels>
    <label>Dieses ist ein manueller Angebot</label>
        <name>quoteManual</name>
    </customLabels>
</Translations>
```

Usage

When you use the `retrieve()` call to get translations in your organization, the files returned in the `.translations` folder only include translations for the other metadata types referenced in `package.xml`. For example, the following `package.xml` file contains `types` elements that match all custom applications, custom labels, Web links defined in home page components, custom tabs, report types, and s-controls. Translations for all these metadata types are returned because each metadata type is explicitly listed in `package.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
       <members>*</members>
       <name>CustomApplication</name>
    </types>
    <types>
       <members>*</members>
       <name>CustomLabels</name>
    </types>
    <types>
       <members>*</members>
       <name>CustomPageWebLink</name>
    </types>
    <types>
       <members>*</members>
       <name>CustomTab</name>
    </types>
    <types>
       <members>*</members>
       <name>ReportType</name>
    </types>
    <types>
       <members>*</members>
       <name>Scontrol</name>
    </types>
    <types>
       <members>*</members>
        <name>Translations</name>
    </types>
    <version>20.0</version>
</Package>
```

See Also:

CustomLabels

Workflow

Represents the metadata associated with a 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. For more information, see “Managing Workflow and Approvals” in the Salesforce.com online help. It extends the Metadata metadata type and inherits its `fullName` field. Use this metadata type to create, update, or delete workflow rule definitions.

When using a manifest file, retrieve all workflow components using the following code:

```
<types>
    <members>*</members>
    <name>Workflow</name>
</types>
```

Declarative Metadata File Suffix and Directory Location

The file suffix is `.workflow` for the workflow file. There is one file per standard or custom object that has workflow, which are stored in the `workflows` directory of the corresponding package.

Version

Workflow rules are available in API version 13.0 and later.

Workflow

This metadata type represents the valid types of workflow rules and actions associated with a standard or custom object.

Field Name	Field Type	Description
`alerts`	WorkflowAlert[]	An array of all alerts for the object associated with the workflow.
`fieldUpdates`	WorkflowFieldUpdate[]	An array of all field updates for the object associated with the workflow.
`fullName`	string	The developer name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`outboundMessages`	WorkflowOutboundMessage[]	An array of all of the outbound messages for the object associated with the workflow.
`rules`	WorkflowRule[]	An array of all the object associated with the workflow.

Field Name	Field Type	Description
`tasks`	WorkflowTask[]	An array of all the tasks for the object associated with the workflow.

WorkflowAlert

WorkflowAlert represents an email alert associated with a workflow rule.

Field Name	Field Type	Description
`ccEmails`	string[]	Additional CC email addresses.
`description`	string	Required. A description of the email alert. Available in API version 16.0 and later.
`fullName`	string	Required. The developer name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`protected`	boolean	Required. Indicates whether this component is protected (`true`) or not (`false`). Protected components cannot be linked to or referenced by components created in the installing organization.
`recipients`	WorkflowEmailRecipient[]	The recipients for the email.
`senderAddress`	string	The address in the From field for the email alert. This allows you to use a standard global email address for your organization (such as `support@company.com`) instead of the default From field, which is the email address of the person who updates the record. You can only specify a value in this field if the `senderType` is set to `OrgWideEmailAddress`. See “Organization-Wide Addresses” in the Salesforce.com online help.
`senderType`	ActionEmailSenderType (enumeration of type string)	The email used as the sender's From and Reply-To addresses. The following values are valid: • `CurrentUser`—The email address of the person updating the record. This is the default setting. • `DefaultWorkflowUser`—The email address of the default workflow user. • `OrgWideEmailAddress`—A verified global email address for your organization, such as `support@company.com`.
`template`	string	Required. Named reference to an EmailTemplate. This email template does not have to exist in the zip file, but it must exist in the Metadata API.

WorkflowEmailRecipient

WorkflowEmailRecipient represents a recipient for an email alert associated with a workflow rule.

Field Name	Field Type	Description
`field`	string	Name of the field referenced in `type`. The field named should be of the type specified in `type`.
`recipient`	string	The recipients for the email. Depending on the type selected, this may be required.
`type`	ActionEmailRecipientTypes (enumeration of type string)	Named reference to an EmailTemplate component. Valid values are: • `accountOwner` - The email is sent to the record's account owner (for example, the Account owner for an Opportunity). • `accountTeam` - Only applicable on the Account object. The email is sent to everyone on that Account's account team. • `campaignMemberDerivedOwner` - Emails are sent to lead and contact owners when contacts are added to a campaign or in response to a campaign. • `contactLookup` - The email is sent to a contact whose value is looked up from a field on the record. For this value, the `field` field must reference a Contact. • `creator` - The email is sent to the record's creator. • `customerPortalOwner` - The email is sent to a specific self-service portal user. For this value, the recipient field must reference a User (by username), only self-service portal users. • `email` - The email is sent to an email address whose value is looked up from a field on the record. For this value, the `field` field must reference an email field. • `group` - The email is sent to all users in a group. For this value, the recipient field must reference a group (by group name). • `opportunityTeam` - Only applicable on the Opportunity object. The email is sent to everyone on that Opportunity's sales team. • `owner` - The email is sent to the record's owner. • `partnerUser` - The email is sent to a specific partner user. For this value, the recipient field must reference a User (by username), only partner users. • `portalRole` - Like `role`, but for portal roles only. • `portalRoleSubordinates` - Like `roleSubordinates`, but for portal roles only. • `role` - The email is sent to all users in a role. For this value, the recipient field must reference a Role (in the role hierarchy, by role name). • `roleSubordinates` - The email is sent to all users in a role subordinates. For this value, the recipient field must reference a Role.

Field Name	Field Type	Description
		• `roleSubordinatesInternal` - Like `roleSubordinates`, but for internal portal roles only • `user` - The email is sent to a specific user. For this value, the recipient field must reference a User (by username) • `userLookup` - The email is sent to a user whose value is looked up from a field on the record. For this value, the `field` field must reference a user foreign key field.

WorkflowFieldUpdate

WorkflowFieldUpdate represents a workflow field update. Field updates allow you to automatically update a field value to one that you specify when a workflow rule is triggered. For more information, see "Defining Field Updates" in the Salesforce.com online help.

Field Name	Field Type	Description
`description`	string	The description of the field update. This information is useful to track the reasoning for initially configuring the field update.
`field`	string	Required. The field (on the object for the workflow) to be updated.
`formula`	string	If the `operation` field value is `Formula`, this is set to a formula used to compute the new field value.
`fullName`	string	Required. The developer name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`literalValue`	string	If the `operation` field value is `Literal`, this is the literal value for the field.
`lookupValue`	string	If the `operation` field value is `lookupValue`, this is the lookup value that is referenced.
`lookupValueType`	LookupValueType (enumeration of type string)	The type of object that the `lookupValue` field value is referencing. The valid values are: • `Queue` • `RecordType` • `User`
`name`	string	Required. A name for the component. Available in version API 16.0 and later.
`notifyAssignee`	boolean	Required. Notify the assignee when the field is updated.

Field Name	Field Type	Description
`operation`	FieldUpdateOperation (enumeration of type string)	Required. The operation that computes the value with which to update the field. Valid values are: • `Formula` - Indicates the field will be set to a formula. If set, the formula must be a valid formula. • `Literal` - Indicates the field will be set to a literal value. If set, the literalValue must be a valid literal value for this field. • `LookupValue` - Similar to Literal, but for an object reference, such as a contact, user, account, etc. If set, the `lookupValue` element must be set. Only User is supported in the current API. • `NextValue` - Indicates the field will be set its next value; this is only allowed when the field update references a picklist. • `Null` - Indicates the field will be set to null. • `PreviousValue` - Indicates the field will be set its previous value; this is only allowed when the field update references a picklist.
`protected`	boolean	Required. Indicates whether this component is protected (`true`) or not (`false`). Protected components cannot be linked to or referenced by components created in the installing organization.
`targetObject`	string	This is set if the change is detected on a child record. If this is set, it points to the foreign key reference on the child object (for example, `EmailMessage.ParentId`) pointing to the parent (for example, Case). When set, the formula is based on the child object (for example, `EmailMessage`). This field is named `sourceField` before version 14.0. The field name change is automatically handled between versions and does not require any manual editing of existing XML component files.

WorkflowOutboundMessage

WorkflowOutboundMessage represents an outbound message associated with a workflow rule. Outbound messages are workflow and approval actions that send the information you specify to an endpoint you designate, such as an external service. An outbound message sends the data in the specified fields in the form of a SOAP message to the endpoint. For more information, see “Defining Outbound Messages” in the Salesforce.com online help.

Field Name	Field Type	Description
`apiVersion`	double	Required. The API version of the outbound message. This is automatically set to the current API version when the outbound message is created. Valid API versions for outbound messages are `8.0` and `18.0` or later. This API version is used in any API calls back to Salesforce.com using the enterprise or partner WSDLs. The `API Version` can only be modified by using the Metadata API. It can't be modified

Field Name	Field Type	Description
		using the Salesforce.com user interface. This field is available in API version 18.0 and later. **Caution:** If you change the `apiVersion` to a version that doesn't support one of the `fields` configured for the outbound message, messages will fail until you update your outbound message listener to consume the updated WSDL. You can monitor the status of outbound messages by viewing ***Your Name*** ➤ **Setup** ➤ **Monitoring** ➤ **Outbound Messages** in Salesforce.com.
`description`	string	Describes the outbound message.
`endpointUrl`	string	Required. The endpoint URL to which the outbound message is sent.
`fields`	string[]	The named references to the field that are to be sent.
`fullName`	string	Required. The developer name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`includeSessionId`	boolean	Required. Set if you want the Salesforce.com *session ID* included in the outbound message. Useful if you intend to make API calls and you do not want to include a username and password.
`integrationUser`	string	Required. The named reference to the user under which this message is sent.
`name`	string	Required. A name for the component. Available in version API 16.0 and later.
`protected`	boolean	Required. Indicates whether this component is protected (`true`) or not (`false`). Protected components cannot be linked to or referenced by components created in the installing organization.
`useDeadLetterQueue`	boolean	This field is only available for organizations with dead letter queue permissions turned on. If set, this outbound message will use the dead letter queue if normal delivery fails.

WorkflowRule

This metadata type represents a workflow rule. It extends the Metadata metadata type and inherits its `fullName` field.

Field Name	Field Type	Description
`actions`	WorkflowActionReference[]	An array of references for the actions that should happen when this rule fires.
`active`	boolean	Required. Determines if this rule is active.
`booleanFilter`	string	For advanced criteria filter, the boolean formula, for example, `(1 AND 2) OR 3`.

Field Name	Field Type	Description
criteriaItems	FilterItem[]	An array of the boolean criteria (conditions) under which this rule fires. Note that either this or formula must be set.
description	string	The description of the workflow rule
formula	string	The formula condition under which this rule first (either this or criteriaItems must be set)
fullName	string	The developer name used as a unique identifier for API access. The fullName can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
triggerType	WorkflowTriggerTypes (enumeration of type string)	Under what conditions the trigger fires. Valid values are: • onAllChanges - The workflow rule is considered on all changes. • onCreateOnly - The workflow rule is considered only on create. • onCreateOrTriggeringUpdate - The workflow rule is considered on create and triggering update.

WorkflowActionReference

WorkflowActionReference represents one of the four workflow actions.

Field Name	Field Type	Description
name	string	Required. The name of the workflow action.
type	WorkflowActionType (enumeration of type string)	Required. There are four types of workflow actions: • Alert • FieldUpdate • OutboundMessage • Task

WorkflowTask

This metadata type references an assigned workflow task.

Field Name	Field Type	Description
assignedTo	string	Specifies the user, role, or team to which the workflow rule or action is assigned. The field corresponding to the value specified here must be the same as the specified assignedToType.

Field Name	Field Type	Description
`assignedToType`	ActionTaskAssignedToTypes (enumeration of type string)	Valid string values for this type are: • `accountCreator` - When set, the task is assigned to the record's account's creator. • `accountOwner` - When set, the task is assigned to the record's account's owner (Opportunity). • `accountTeam` - Same as WorkflowAlert type • `creator` - When set, the task is assigned to the record's creator. • `opportunityTeam` - Same as WorkflowAlert type • `owner` - When set, the task is assigned to the record's owner. • `partnerUser` - When set, the `assignedTo` field references a User (by username), a partner user. • `portalRole` - When set, the `assignedTo` field references a Role (by role name), a portal role. • `role` - When set, the `assignedTo` field references a Role (by role name) • `user` - When set, the `assignedTo` field references a User (by username)
`description`	string	The description of this workflow task.
`dueDateOffset`	int	Required. The offset, in days, from either the trigger date, or the date specified in the (optional) `offsetFromField`. This can be a negative number.
`fullName`	string	Required. The developer name used as a unique identifier for API access. The `fullName` can contain only underscores and alphanumeric characters. It must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is inherited from the Metadata component.
`notifyAssignee`	boolean	Required. Set to send an email notification when the task is assigned.
`offsetFromField`	string	Optional field reference of the date field from which the `dueDate` should be computed.
`priority`	string	Required. The priority to assign the created task.
`protected`	boolean	Required. Indicates whether this component is protected (`true`) or not (`false`). Protected components cannot be linked to or referenced by components created in the installing organization.
`status`	string	Required. The status to assign the created task.
`subject`	string	Required. A subject for the workflow task. It is used if an email notification is sent when the task is assigned. Available in API version 16.0 and later.

Declarative Metadata Sample Definition

The following is the definition of a workflow rule:

```
<?xml version="1.0" encoding="UTF-8"?>
<Workflow xmlns="http://soap.sforce.com/2006/04/metadata">
    <alerts>
        <fullName>Another_alert</fullName>
        <description>Another alert</description>
        <protected>false</protected>
        <recipients>
            <type>accountOwner</type>
        </recipients>
        <recipients>
            <field>Contact__c</field>
            <type>contactLookup</type>
        </recipients>
        <recipients>
            <field>Email__c</field>
            <type>email</type>
        </recipients>
        <template>TestEmail/Email Test</template>
    </alerts>
    <fieldUpdates>
        <fullName>Enum_Field_Update</fullName>
        <description>Blah</description>
        <field>EnumField__c</field>
        <name>Enum Field Update</name>
        <notifyAssignee>true</notifyAssignee>
        <operation>NextValue</operation>
        <protected>false</protected>
    </fieldUpdates>
    <fieldUpdates>
        <fullName>Enum_Field_Update2</fullName>
        <description>Blah</description>
        <field>EnumField__c</field>
        <literalValue>PLX2</literalValue>
        <name>Enum Field Update2</name>
        <notifyAssignee>true</notifyAssignee>
        <operation>Literal</operation>
        <protected>false</protected>
    </fieldUpdates>
    <fieldUpdates>
        <fullName>Field_Update</fullName>
        <description>TestField update desc</description>
        <field>Name</field>
        <formula>Name &amp; &quot;Updated&quot;</formula>
        <name>Field Update</name>
        <notifyAssignee>false</notifyAssignee>
        <operation>Formula</operation>
        <protected>false</protected>
    </fieldUpdates>
    <fieldUpdates>
        <fullName>Lookup_On_Contact</fullName>
        <field>RealOwner__c</field>
        <lookupValue>admin@acme.com</lookupValue>
        <name>Lookup On Contact</name>
        <notifyAssignee>false</notifyAssignee>
        <operation>LookupValue</operation>
        <protected>false</protected>
    </fieldUpdates>
    <outboundMessages>
        <fullName>Another_Outbound_message</fullName>
        <description>Another Random outbound.</description>
        <endpointUrl>http://www.test.com</endpointUrl>
        <fields>Email__c</fields>
        <fields>Id</fields>
        <fields>Name</fields>
        <includeSessionId>true</includeSessionId>
        <integrationUser>admin@acme.com</integrationUser>
        <name>Another Outbound message</name>
        <protected>false</protected>
    </outboundMessages>
    <rules>
        <fullName>BooleanFilter</fullName>
        <active>false</active>
        <booleanFilter>1 AND 2 OR 3</booleanFilter>
        <criteriaItems>
            <field>CustomObjectForWorkflow__c.CreatedById</field>
            <operation>notEqual</operation>
        </criteriaItems>
        <criteriaItems>
            <field>CustomObjectForWorkflow__c.CreatedById</field>
            <operation>notEqual</operation>
            <value>abc</value>
        </criteriaItems>
        <criteriaItems>
            <field>CustomObjectForWorkflow__c.CreatedById</field>
            <operation>equals</operation>
            <value>xyz</value>
        </criteriaItems>
        <triggerType>onCreateOrTriggeringUpdate</triggerType>
    </rules>
    <rules>
        <fullName>Custom Rule1</fullName>
        <actions>
            <name>Another_alert</name>
            <type>Alert</type>
        </actions>
        <actions>
            <name>Enum_Field_Update2</name>
            <type>FieldUpdate</type>
        </actions>
        <actions>
            <fullName>Field_Update</name>
                <type>FieldUpdate</type>
        </actions>
        <actions>
            <name>Another_Outbound_message</name>
            <type>OutboundMessage</type>
        </actions>
        <actions>
            <name>Role_task_was_completed</name>
            <type>Task</type>
        </actions>
        <active>true</active>
        <criteriaItems>
            <field>CustomObjectForWorkflow__c.Name</field>
            <operation>startsWith</operation>
            <value>ABC</value>
        </criteriaItems>
        <description>Custom Rule1 desc</description>
        <triggerType>onCreateOrTriggeringUpdate</triggerType>
    </rules>
    <rules>
        <fullName>IsChangedFunctionRule</fullName>
        <active>true</active>
        <description>IsChangedDesc</description>
        <formula>ISCHANGED(Name)</formula>
        <triggerType>onAllChanges</triggerType>
    </rules>
    <tasks>
        <fullName>Another_task_was_completed</fullName>
        <assignedToType>owner</assignedToType>
        <description>Random Comment</description>
        <dueDateOffset>20</dueDateOffset>
        <notifyAssignee>true</notifyAssignee>
        <priority>High</priority>
        <protected>false</protected>
        <status>Completed</status>
        <subject>Another task was completed</subject>
    </tasks>
    <tasks>
        <fullName>Role_task_was_completed</fullName>
        <assignedTo>R11</assignedTo>
        <assignedToType>role</assignedToType>
        <dueDateOffset>-2</dueDateOffset>
        <notifyAssignee>true</notifyAssignee>
        <offsetFromField>CustomObjectForWorkflow__c.CreatedDate</offsetFromField>
        <priority>High</priority>
        <protected>false</protected>
        <status>Completed</status>
        <subject>Role task was completed</subject>
    </tasks>
    <tasks>
        <fullName>User_task_was_completed</fullName>
        <assignedTo>admin@acme.com</assignedTo>
        <assignedToType>user</assignedToType>
        <dueDateOffset>-2</dueDateOffset>
        <notifyAssignee>true</notifyAssignee>
        <offsetFromField>User.CreatedDate</offsetFromField>
        <priority>High</priority>
        <protected>false</protected>
        <status>Completed</status>
        <subject>User task was completed</subject>
    </tasks>
</Workflow>
```

Chapter 10

Error Handling

The Metadata API calls return error information that your client application can use to identify and resolve runtime errors. The Metadata API provides the following types of error handling:

- Since the Metadata API uses the enterprise or partner WSDLs to authenticate, it uses SOAP fault messages defined in those WSDLs for errors resulting from badly formed messages, failed authentication, or similar problems. Each SOAP fault has an associated ExceptionCode. For more details, see "Error Handling" in the *Web Services API Developer's Guide*.
- For errors with `create()`, `update()`, and `delete()`, see the error status code in the `statusCode` field in the AsyncResult object for the associated component. For sample code, see Java Sample Code for CRUD-Based Development on page 9.
- For errors with `deploy()`, see the `problem` and `success` fields in the DeployMessage object for the associated component. For sample code, see Java Sample Code for File-Based Development on page 9.
- For errors with `retrieve()`, see the `problem` field in the RetrieveMessage object for the associated component. For sample code, see Java Sample Code for File-Based Development on page 9.

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 automatically expire after the amount of time specified in the **Security Controls** setup area of the Salesforce.com application (default two hours). When your session expires, the exception code INVALID_SESSION_ID is returned. If this happens, you must invoke the `login()` call again. For more information about `login()`, see the *Web Services API Developer's 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

Apex

Force.com Apex code is a strongly-typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Force.com platform server in conjunction with calls to the Force.com API. Using syntax that looks like Java and acts like database stored procedures, Apex code enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex scripts 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.com provides standard apps such as Sales and Call Center. You can customize the standard apps to match the way you work. In addition, you can package an app and upload it to AppExchange along with related components such as custom fields, custom tabs, and custom objects. Then, you can make the app available to other Salesforce.com users from AppExchange.

AppExchange

AppExchange is a sharing interface from salesforce.com that allows you to browse and share apps and services for the Force.com 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 in order to allow other computer programs to request services from it and exchange data between them.

Asynchronous Calls

A call that does not return results immediately because the operation may take a long time. Calls in the Metadata API and Bulk API are asynchronous.

B

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

The REST-based Bulk API is optimized for processing large sets of data. It allows you to insert, update, upsert, or delete a large number of records asynchronously by submitting a number of batches which are processed in the background by Salesforce.com. See also Web Services API.

C

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.com user interface and uses only the Force.com API or Bulk API. It typically runs on a desktop or mobile device. These apps treat the platform as a data source, using the development model of whatever tool and platform for which they are designed. See also Composite App and Native App.

Component, Metadata

A component is an instance of a metadata type in the Metadata API. For example, CustomObject is a metadata type for custom objects, and the `MyCustomObject__c` component is an instance of a custom object. A component is described in an XML file and it can be deployed or retrieved using the Metadata API, or tools built on top of it, such as the Force.com IDE or the Force.com Migration Tool.

Component, Visualforce

Something that can be added to a Visualforce page with a set of tags, for example, `<apex:detail>`. Visualforce includes a number of standard components, or you can create your own custom components.

Component Reference, Visualforce

A description of the standard and custom Visualforce components that are available in your organization. You can access the component library from the development footer of any Visualforce page or the *Visualforce Developer's Guide*.

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.com 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 URLs defined by administrators to integrate your Salesforce.com 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 Force.com platform includes a database where your data is stored.

Database Table

A list of information, presented with rows and columns, about the person, thing, or concept you want to track. See also Object.

Data Manipulation Language (DML)

An Apex method or operation that inserts, updates, or deletes records from the Force.com platform database.

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.

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.com organization designed for developers to extend, integrate, and develop with the Force.com platform. Developer Edition accounts are available on developer.force.com.

Developer Force

The Developer Force website at developer.force.com provides a full range of resources for platform developers, including sample code, toolkits, an online developer community, and the ability to obtain limited Force.com platform environments.

Document Library

A place to store documents without attaching them to accounts, contacts, opportunities, or other records.

E

Email Alert

Email alerts are workflow and approval actions that are generated using an email template by a workflow rule or approval process and sent to designated recipients, either Salesforce.com users or others.

Email Template

A form email that communicates a standard message, such as a welcome letter to new employees or an acknowledgement that a customer service request has been received. Email templates can be personalized with merge fields, and can be written in text, HTML, or custom format.

Enterprise Edition

A Salesforce.com edition designed for larger, more complex businesses.

Enterprise WSDL

A strongly-typed WSDL for customers who want to build an integration with their Salesforce.com organization only, or for partners who are using tools like Tibco or webMethods to build integrations that require strong typecasting. The downside of the Enterprise WSDL is that it only works with the schema of a single Salesforce.com organization because it is bound to all of the unique objects and fields that exist in that organization's data model.

Entity Relationship Diagram (ERD)

A data modeling tool that helps you organize your data into entities (or objects, as they are called in the Force.com platform) and define the relationships between them. ERD diagrams for key Salesforce.com objects are published in the *Web Services API Developer's Guide*.

Enumeration Field

An enumeration is the WSDL equivalent of a picklist field. The valid values of the field are restricted to a strict set of possible values, all having the same data type.

F

Field

A part of an object that holds a specific piece of information, such as a text or currency value.

Field-Level Security

Settings that determine whether fields are hidden, visible, read only, or editable for users based on their profiles. Available in Enterprise, Unlimited, and Developer Editions only.

Filter Condition/Criteria

Condition on particular fields that qualifies items to be included in a list view or report, such as “State equals California.”

Force.com

The salesforce.com platform for building applications in the cloud. Force.com combines a powerful user interface, operating system, and database to allow you to customize and deploy applications in the cloud for your entire enterprise.

Force.com IDE

An Eclipse plug-in that allows developers to manage, author, debug and deploy Force.com applications in the Eclipse development environment.

Force.com Migration Tool

A toolkit that allows you to write an Apache Ant build script for migrating Force.com components between a local file system and a Salesforce.com organization.

Foreign key

A field whose value is the same as the primary key of another table. You can think of a foreign key as a copy of a primary key from another table. A relationship is made between two tables by matching the values of the foreign key in one table with the values of the primary key in another.

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 twelve 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.com 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 Force.com platform runs on multiple instances, but data for any single organization is always consolidated on a single instance.

Integration User

A Salesforce.com user defined solely for client apps or integrations. Also referred to as the logged-in user in a Web services API context.

ISO Code

The International Organization for Standardization country code, which represents each country by two letters.

J

Junction Object

A custom object with two master-detail relationships. Using a custom junction object, you can model a "many-to-many" relationship between two objects. For example, you may have a custom object called "Bug" that relates to the standard case object such that a bug could be related to multiple cases and a case could also be related to multiple bugs.

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 a managed package of yours from AppExchange.

License Management Organization (LMO)

The Salesforce.com organization that you use to track all the Salesforce.com 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, or Developer Edition organization as your license management organization. For more information, go to http://www.salesforce.com/docs/en/lma/index.htm.

List View

A list display of items (for example, accounts or contacts) based on specific criteria. Salesforce.com provides some predefined views.

In the Console tab, the list view is the top frame that displays a list view of records based on specific criteria. The list views you can select to display in the console are the same list views defined on the tabs of other objects. You cannot create a list view within the console.

Local Project

A `.zip` file containing a project manifest (`package.xml` file) and one or more metadata components.

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, 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 Web services API context, the username used to log into Salesforce.com. Client applications run with the permissions and sharing of the logged-in user. Also referred to as an integration user.

Lookup Field

A type of field that contains a linkable value to another record. You can display lookup fields on page layouts where the object has a lookup or master-detail relationship with another object. For example, cases have a lookup relationship with assets that allows users to select an asset using a lookup dialog from the case edit page and click the name of the asset from the case detail page.

M

Managed Package

A collection of application components that are posted as a unit on AppExchange, and are associated with a namespace and possibly a License Management Organization. A package must be managed for it to support upgrades. An organization can create a single managed package that can be downloaded and installed by many different organizations. They differ from unmanaged packages in that some components are locked, 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, so as to protect the intellectual property of the developer.

Manifest File

The project manifest file (`package.xml`) lists the XML components to retrieve or deploy when working with the Metadata API, or clients built on top of the Metadata API, such as the Force.com IDE or the Force.com Migration Tool.

Manual Sharing

Record-level access rules that allow record owners to give read and edit permissions to other users who might not have access to the record any other way.

Many-to-Many Relationship

A relationship where each side of the relationship can have many children on the other side. Many-to-many relationships are implemented through the use of junction objects.

Master-Detail Relationship

A relationship between two different types of records that associates the records with each other. For example, accounts have a master-detail relationship with opportunities. This type of relationship affects record deletion, security, and makes the lookup relationship field required on the page layout.

Metadata

Information about the structure, appearance, and functionality of an organization and any of its parts. Force.com uses XML to describe metadata.

Metadata WSDL

A WSDL for users who want to use the Force.com Metadata API calls.

Multitenancy

An application model where all users and apps share a single, common infrastructure and code base.

N

Namespace

In a packaging context, a one- to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other developers onAppExchange, similar to a domain name. Salesforce.com automatically prepends your namespace prefix, followed by two underscores ("__"), to all unique component names in your Salesforce.com organization.

Native App

An app that is built exclusively with setup (metadata) configuration on Force.com. Native apps do not require any external services or infrastructure.

O

Object

An object allows you to store information in your Salesforce.com 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 tabs and objects from a user so that he or she does not know that type of data exists. On the platform you set object-level access rules with object permissions on user profiles.

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 make it so that any user can see any record of a particular object that is enabled in their user profile, but that they need extra permissions to edit one.

Outbound Message

An outbound message is a workflow, approval, or milestone action that sends the information you specify to an endpoint you designate, such as an external service. An outbound message sends the data in the specified fields in the form of a SOAP message to the endpoint. Outbound messaging is configured in the Salesforce.com setup menu. Then you must configure the external endpoint. You can create a listener for the messages using the Web services 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

Package

A group of Force.com components and applications that are made available to other organizations through the AppExchange. You use packages to bundle an app along with any related components so that you can upload them to AppExchange together.

Parent Account

Organization or company that an account is affiliated with or owned by. 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.com organizations. With this WSDL, the developer is responsible for marshaling data in the correct object representation, which typically involves editing the XML. However, the developer is also freed from being dependent on any particular data model or Salesforce.com organization. Contrast this with the Enterprise WSDL, which is strongly typed.

Personal Edition

Product designed for individual sales representatives and single users.

Picklist

Selection list of options available for specific fields in a Salesforce.com 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.com 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 Edition

A Salesforce.com edition based on either Enterprise Edition or Unlimited Edition that does not include any of the standard Salesforce.com CRM apps, such as Sales or Service & Support.

Primary Key

A relational database concept. Each table in a relational database has a field in which the data value uniquely identifies the record. This field is called the primary key. The relationship is made between two tables by matching the values of the foreign key in one table with the values of the primary key in another.

Production Organization

A Salesforce.com organization that has live users accessing data.

Professional Edition

A Salesforce.com edition designed for businesses who need full-featured CRM functionality.

Q

Queue

A holding area for items before they are processed. Salesforce.com uses queues in a number of different features and technologies.

Query String Parameter

A name-value pair that's included in a URL, typically after a '?' character. For example:

```
http://na1.salesforce.com/001/e?name=value
```

R

Record

A single instance of a Salesforce.com object. For example, "John Jones" might be the name of a contact record.

Record Name

A standard field on all Salesforce.com objects. Whenever a record name is displayed in a Force.com application, the value is represented as a link to a detail view of the record. A record name can be either free-form text or an autonumber field. `Record Name` does not have to be a unique value.

Record Type

A field available for certain records that can include some or all of the standard and custom picklist values for that record. Record types are special fields that you can associate 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 by using the link in the sidebar.

Related Object

Objects chosen by an administrator to display in the Console tab'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.com 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.com SOA (Service-Oriented Architecture)

A powerful capability of Force.com that allows you to make calls to external Web services from within Apex.

Sandbox Organization

A nearly identical copy of a Salesforce.com production organization. You can create multiple sandboxes in separate environments for a variety of purposes, such as testing and training, without compromising the data and applications in your production environment.

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.

Session ID

An authentication token that is returned when a user successfully logs in to Salesforce.com. The Session ID prevents a user from having to log in again every time he or she wants to perform another action in Salesforce.com. Different from a record ID or Salesforce.com ID, which are terms for the unique ID of a Salesforce.com record.

Session Timeout

The period of time after login before a user is automatically logged out. Sessions expire automatically after a predetermined length of inactivity, which can be configured in Salesforce.com by clicking ***Your Name*** ➤ **Setup** ➤ **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

An administration area where you can customize and define Force.com applications. Access Setup through the ***Your Name*** ➤ **Setup** link at the top of Salesforce.com pages.

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 a specific account or opportunity with other users or groups.
- Apex-Managed Sharing—enables developers to programmatically manipulate sharing to support their application's behavior. See Apex-Managed Sharing.

Sharing Model

Behavior defined by your administrator that determines default access by users to different types of records.

Sharing Rule

Type of default sharing created by administrators. Allows users in a specified group or role to have access to all information created by users within a given group or role.

Sites

Force.com sites enables you to create public websites and applications that are directly integrated with your Salesforce.com 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.

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.com Object Query Language)

A query language that allows you to construct simple but powerful query strings and to specify the criteria that should be used to select data from the Force.com database.

SOSL (Salesforce.com Object Search Language)

A query language that allows you to perform text-based searches using the Force.com API.

Standard Object

A built-in object included with the Force.com platform. You can also build custom objects to store information that is unique to your app.

System Log

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 Force.com IDE.

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.com 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 all of Salesforce.com 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.com's flagship solution for maximizing CRM success and extending that success across the entire enterprise through the Force.com platform.

Unmanaged Package

A package that cannot be upgraded or controlled by its developer.

URL (Uniform Resource Locator)

The global address of a website, document, or other resource on the Internet. For example, http://www.salesforce.com.

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.com 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.

WebService Method

An Apex class method or variable that can be used by external systems, like a mash-up with a third-party application. Web service methods must be defined in a global class.

Web Services API

A SOAP-based Web services application programming interface that provides access to your Salesforce.com organization's information. See also Bulk API.

Web Tab

A custom tab that allows your users to use external websites from within the application.

Workflow and Approval Actions

Workflow and approval actions consist of email alerts, tasks, field updates, and outbound messages that can be triggered by a workflow rule or approval process.

Workflow Action

An email alert, field update, outbound message, or task that 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.

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.com Enterprise WSDL or Partner WSDL to communicate with Salesforce.com using the Web services API.

X

XML (Extensible Markup Language)

A markup language that enables the sharing and transportation of structured data. All Force.com components that are retrieved or deployed through the Metadata API are represented by XML definitions.

Y

No Glossary items for this entry.

Z

Zip File

A data compression and archive format.

A collection of files retrieved or deployed by the Metadata API. See also Local Project.

Index

A

ActionOverride component 94
AnalyticSnapshot component 72
ApexClass component 80
ApexComponent component 83
ApexPage component 84
ApexTrigger component 85
ArticleType component 74, 78
    Layout 77
Asterisk wildcard 25

B

BusinessProcess component 96

C

Calls
    checkDeployStatus 41
    checkRetrieveStatus 50
    checkStatus 57
    create (asynchronous) 51
    delete (asynchronous) 52
    deploy 34
    describeMetadata 58
    listMetadata 59, 60
    retrieve 42
    update (asynchronous) 54
checkDeployStatus metadata call 41
checkRetrieveStatus metadata call 50
checkStatus metadata call 57
components
    DataCategoryGroup 148
Components
    ActionOverride 94
    ApexClass 80
    ApexComponent 83
    ApexPage 84
    ApexTrigger 85
    Article Type 78
    ArticleType 74
    BusinessProcess 96
    CustomApplication 87
    CustomField 97
    CustomFieldTranslation 125
    CustomLabel 88
    CustomLabels 88
    CustomObject 89
    CustomObjectTranslation 125
    CustomPageWebLink 131
    CustomTab 137
    Dashboard 139
    Document 153
    EmailTemplate 155
    Folder 158
    HomePageComponent 160
    HomePageLayout 161
    Layout 162
    Layout (for article types) 77
    Letterhead 168
Components *(continued)*
    ListView 103
    ListViewFilter 103
    Metadata 170
    MetadataWithContent 171
    NamedFilter 106
    ObjectRelationship 199
    Picklist 108
    Profile 174
    RecordType 114
    ReportType 199
    Scontrol 202
    SearchLayouts 115
    SharingReason 118
    SharingRecalculation 119
    Static resource 204
    Translations 205
    ValidationRule 119
    Workflow 210
create call (asynchronous) 51
Custom report type component 199
CustomApplication component 87
CustomField component 97
CustomLabels component 88
CustomObject
    Weblink component 120
CustomObject component 89
CustomObjectTranslation component 125
CustomPageWebLink component 131
CustomSite component 134
CustomTab component 137

D

Dashboard component 139
DataCategoryGroup component 148
Declarative development 25
delete call (asynchronous) 52
deploy call 34
    running tests 31
describeMetadata call 58
Document component 153

E

Editions 6
EmailTemplate component 155
EntitlementTemplate component 157
Error handling 221
Expiration of session ID 221

F

Field types 124
File-based metadata 25
Folder component 158

H

HomePageComponent component 160
HomePageLayout component 161

L

Layout component 162
Layout component (for article types) 77
Letterhead component 168
listMetadata call 59
ListMetadataQuery 60
ListView component 103

M

Manifest file 25, 29
Metadata 25
Metadata calls 5
Metadata component 170, 171
Metadata components 71
  AnalyticSnapshot 72
  CustomSite 134
  EntitlementTemplate 157
  Portal 172
  RemoteSiteSetting 182
  Report 183
Metadata types 71

N

NamedFilter component 106

O

Outer join 199

P

Package versions 80
package.xml 25
  samples 29
PackageVersion 80
Picklist definitions 108
Portal component 172
Profile component 174

Q

Quick start 9

R

RecordType component 114
Recycle Bin 153
RemoteSiteSetting component 182
Report component 183
retrieve call 42
RetrieveRequest 48

S

Sample code 9
Sandbox 6
Scontrol component 202
SearchLayouts component 115
Session ID expiration 221
SharingReason component 118
SharingRecalculation component 119
Static resource component 204
Support policy
  Backward compatibility 7
  call deprecation 7
  Deprecated calls 7
  Metadata API 7
Supported calls 71

T

Translations component 205
Types of fields 124

U

Understanding metadata calls and components
  metadata 5
update call (asynchronous) 54

V

ValidationRule component 119
Versions 80
Visualforce component, see ApexComponent 83
Visualforce page, see ApexPage 84

W

Weblink component of CustomObject 120
Workflow component 210
WSDL integration 9

Z

Zip file 25